Designing a Raster-Image Processor

The road from screen display to hard copy is circuitous

Jon Barrett and Kirk Reistroffer

ASKING THE QUESTION "How do you drive a laser printer?" is less like "How do you drive a car?" than like "How do you drive to Cambridge?" The question "How do you drive a car?" has a fairly specific and uniform answer having to do with using a steering wheel and two or three floor pedals. But "How do you drive to Cambridge?" depends on where you are starting from and whether you want to take the fast or the scenic route. Similarly, "How do you drive a laser printer?" depends upon where you are starting from and what kind of results you expect.

Despite this, some generalizations can be made about the aims and methods of driving laser printers. In this article we will talk about a generalized front end for a raster-image processor (RIP) and explain how what shows up on-screen finally shows up on a laser-printer output. First we will describe some of the page-description languages (PDLs) important in the desktop-publishing world. Then we will explain how RIPs work in general. Finally, we will discuss our approach to the hardware and software necessary to drive a laser printer, with emphasis on what we think are some innovations in the technology.

In our generalized example of an electronic publishing system (EPS), we have several aims. First, we want it to show on-screen what will appear on paper. That is, we want a WYSIWYG (what you see is what you get) system. Second, it must generate sophisticated graphics, allow easy revision of them, and display them on-screen. Third, it must allow for mixing typefaces and sizes on any one page, preferably without limit. Fourth, it must allow mixing of graphics and text, preferably without any limit on complexity. Fifth, it must be able to print out, with fidelity, all that the user has created. Sixth, the quicker it can do this, the better.

To accomplish these aims, we need several components. The front end on the host computer must let you create documents of arbitrary graphic and typographic complexity. Second, documents must be translated into a PDL, a series of commands which a laser printer can "understand." Finally, the system must have hardware that takes the digital information describing the page and turns it into modulations of the laser, causing the image to appear on the paper. This hardware is called a RIP. (A raster, or raster scan image, as used in the context of computer graphics, is a two-dimensional array of pixels.)

It might seem that the process of getting the document's image onto the CRT screen is essentially the same, from the logical point of view, as getting it onto paper. To a certain extent, that is correct. The image on paper and the CRT image are both views into the same database, in this case a document. But several issues arise that distinguish the two processes. Chief among these are the questions of hardware and resolution dependence. Most EPSs have opted for some form of independence in both regards. That is one reason why PostScript is so popular; a PostScript description of a page will theoretically produce the same page, given differences in resolution, on any PostScript output device. Thus, EPS software designers can write a single driver for their system and have confidence that it will produce accurate printed pages when hooked up to a variety of printers. The chief disadvantage is that this dependence prevents an EPS from taking full advantage of a specific output device.

In fact, as we shall explain, in our design we opted for some limited device and resolution dependence. This means that we do not have to pass parameters with enough information to drive a typesetter at 2000 dots per inch if we are in fact driving a 300-dpi laser printer. This optimizes for data compression, which can improve the printer's total throughput. This is especially true when the communication channel is relatively slow, as in the case of the 9600-bit-per-second serial AppleTalk line used by Apple's LaserWriter. Obviously, a resolution-dependent system demands more initial work from the designers. Also, our device dependence enables us to hand-tune fonts for the particular printers we support, optimizing them for legibility and style.

The question of machine dependence is not academic. For example, some laser printers write to white and some write to black. That is, some consider the image

continued

Jon Barrett (Zedex, 215 First St., Cambridge, MA 02142) was formerly manager of hardware development at Interleaf. Kirk Reistroffer is a senior software engineer at Interleaf (Ten Canal Park, Cambridge, MA 02141).

drum in the printer to be a black object that must be "erased" to carve out black letters, and others consider it to be a white object onto which black dots must be placed. In either case, you want to lay down dots that are slightly larger than a pixel so they will overlap. If you are writing to black (laying down black dots), the character so constructed will be noticeably fatter than one carved out of a black background (see figure 1). A font that works well on one printer can look ungainly on another. Therefore, there is a real reason to use machine-dependent drivers.

But to understand how RIPs and PDLs have evolved the way they have, it is necessary to know a little about the short history of laser printers.

History of Laser Printers

Although laser printers have been around since the mid-1970s, they were text-oriented and in the $500,000 price range. Thus, when Canon introduced the LBP-10 laser printer in 1979 for $10,000, it caused a great deal of excitement.

The first laser printers using the Canon engine, while exciting, were inadequate for the needs of the generalized EPS that is our example. They did support downloaded fonts, but they did not have sufficient graphics capabilities. For example, they could not fill an arbitrary polygon.

Then, in 1981, Imagen introduced a laser printer with an important advance. To compose an entire page before printing, you need enough memory to hold data about each of the dots to be printed, about a megabyte's worth for a 300-dpi printer. At that time, a megabyte was expensive. So Imagen figured out a way to print full pages without having the entire page in memory. This technique was implemented in their RIP. The Imagen technique was clever. It internally considered a page to consist of a matrix of "postage stamps," each of which it could compose in real time. If a "stamp" were text only, it would create the raster image on the fly. If the data were part of a large, uniform area, it would take one "stamp" and replicate it, saving memory.

This was an improvement over the Xerox laser printers of the time, for while they could print as much text as wanted on a page, virtually the only graphics they would print were horizontal and vertical lines for tables. But the early Imagen laser printer could not print pages with very complex graphics. And it did not do clipping. (To clip an image is to crop off selected portions of it.) Also, the amount of memory available too severely limited the number of vector lines that could be printed. (Vector-line graphics is a method for drawing graphics using straight lines. Each line is usually described by its end points.) This is an important limitation when your set of geometric primitives does not include circles and arcs, so curves have to be represented by hundreds or even thousands of vector lines. Finally, because the letters were logically ORed into the graphic "stamps," the printer could not produce white text on a black background: Assuming that a set bit is one that prints, ORing a set and a reset bit results in a set bit.

Previously available laser printers, such as the Hewlett-Packard LaserJet, were very popular although not the most functional. They could handle text adequately but did not shine when it came to complex graphics. The latest generation of laser printers from Hewlett-Packard, Imagen, and other vendors has overcome many of the limitations of earlier models, in large part because memory has become cheap enough to make large memory boards that can hold a full-page bit map. The memory generally consists of at least 1.5 megabytes. Of this, about 1 megabyte is for the page bit map, and the other 0.5 megabyte is for temporary storage, variables, and cached fonts. (Cached fonts are, in essence, fonts downloaded from the host computer or computed from outlines.)

In addition, ROMs are available that may contain font bit maps that the printer will use to compose the text on a page whenever the PDL has told it to insert some characters. The ROM's bit map is compatible with the printer's resolution if the PDL is device-independent. Most printers in this class offer between 1.5 and 2 megabytes of memory. Improved software has enabled these printers to produce pages of considerable complexity.

Putting the Image on Screen and Paper

The main section of an EPS program, which puts the characters on the display screen, is fairly standard. The basic display loop for our generalized EPS has to grab the bits that compose the character and put them into an arbitrary place in memory, typically a screen buffer dedicated to the task. To draw the character quickly, you need to grab the largest chunk of data you can at a time. Our imaginary computer uses the Motorola 68000 that has a 16-bit data bus. Thus, the most data available for grabbing in any one cycle is 16 bits. These bits are then moved into a two-dimensional array.

This would be fairly straightforward if you were dealing with monospaced text, since each character would then occupy a bit map of the same size. But our generalized EPS does not. It provides on-screen proportional spacing. Further, there is the problem of kerning letter pairs in which one character hangs over another as in the case of the letters VA or italicized lower case f's (see figure 2). (Some systems treat some commonly kerned pairs, such as ff, as single bit maps. In traditional typography, such pairs, occupying a single piece of type, are called ligatures.) To achieve this, you cannot

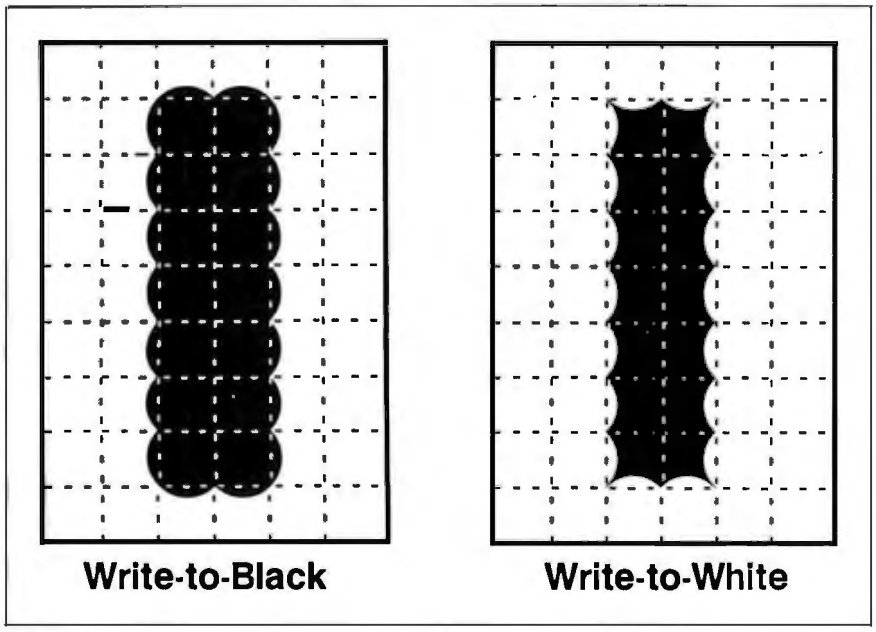

Figure 1: *With write-to-black printers, black pixels are written to a white background. With write-to-white printers, white pixels are written to a black background. The resulting fonts can appear substantially different, as in the case of the sans serif l's shown here. Hand-tuning fonts for specific printers can enhance legibility and style.*

treat each character as occupying an inviolable rectangle. Rather, you must logically OR bit maps that intersect. When you OR them, all bits currently set in the destination bit map remain set, while those set in the intersecting source bit map next set the appropriate bits in the destination bit map.

This solves the problem of displaying bit-mapped, proportionally spaced characters on-screen. But the 300-dpi resolution of a laser printer is four times greater than the approximately 75-dpi resolution of workstation display screens, which means the printer has 16 times the number of dots per page. Although the algorithm is basically the same, because it is 16 times the data, the job is slower.

The problem is how to blit, that is move or logically OR a source bit map of a glyph (i.e., a bit map that contains a character or a symbol, typically one that will be reused) into the destination bit map (screen or printer). ("Blit" as a verb comes from BLT, an acronym for bit boundary block transfer.) As an example, let's take a simple case such as blitting the letter A. If the source is an array one word wide, we want to write this to a destination that is also one word wide. But remember that we are dealing with proportionally spaced text. Thus, glyphs will not necessarily fall on word boundaries in the destination. With a 16-bit word (which is the size of the word we actually deal with), the chances are only 1 in 16 that the glyph will fall on the boundary. Thus, our problem is writing a glyph that is one word wide to a destination where it will straddle two words. (In general, if n is the width of the glyph in words, the problem will be to write it to a destination $n+1$ words wide.)

The basic loop for a software blit (see figure 3) consists of the following steps: First, you get the source word, then shift it right to reflect the offset from the destination's word boundary. Next, get the destination word. OR the shifted source and destination word, and write it to the destination. Get the source word again (or keep it in a register), then shift it left to reflect the offset. Get the destination +1 word. OR the shifted source and destination +1, and write to destination +1.

The second access of the source word is necessary to take care of any portions of the source bit map that run over into a second word space.

Document to PDL

We now have some characters on the screen. How do we get them to the printer? Remember that what is written to screen and what is written to the printer are two views of the same document. This document, at least the text portion of it, is one-dimensional. In putting it on the screen, the EPS software has to make decisions about how to bring it into two dimensions: where should the line breaks

continued

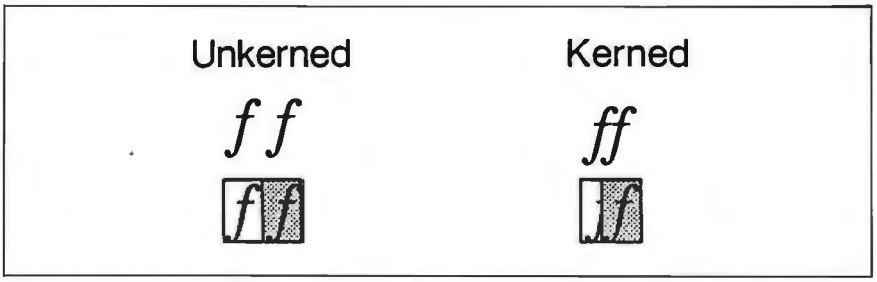

Figure 2: *Unkerned italicized f's, shown on the left, each occupy an inviolate rectangle. Kerned letters, on the right, overhang each other, complicating the bit-mapping algorithms.*

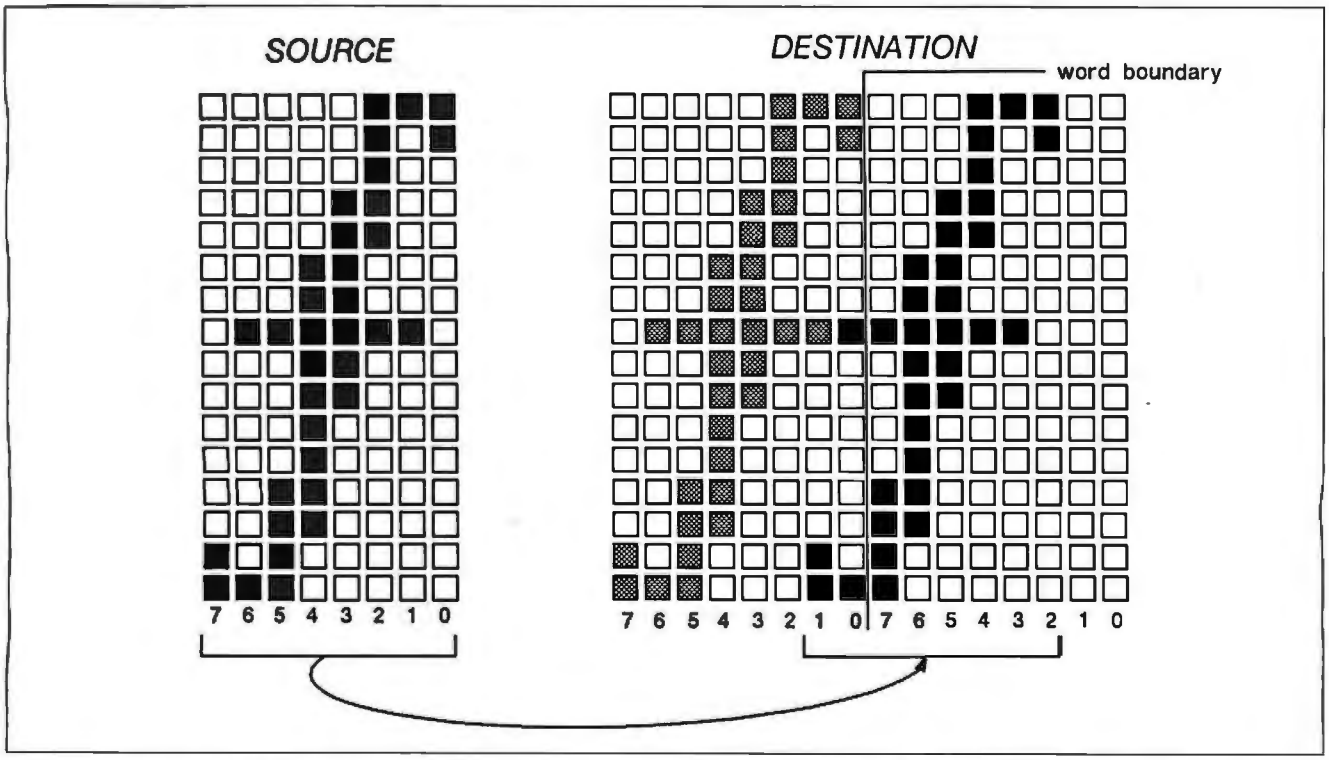

Figure 3: *To move the source bit map row by row into the destination bit map (which can straddle word boundaries), each row must be shifted by the offset from the word boundary. In this case, the offset is six. In this example, kerned italicized f's are shown, but the problem can occur with any letter. This figure uses 8-bit words to simplify the diagram.*

be, where should the page breaks be. In non-WYSIWYG systems, such as batch systems, those decisions are made after all editing has been completed. In a WYSIWYG system, those decisions have to be made on the fly to keep the screen updated. This can be a daunting task, since inserting a single letter while editing can cause a sentence to drop onto the next page, setting off pagination ripples throughout the document. Further, some EPSs (such as Interleaf) provide on-screen hyphenation, which means that after you type each word-terminating character, you have to check the preceding character string against a dictionary to find the hyphenation points. (You can use an algorithm in lieu of a dictionary.)

In a true WYSIWYG system, those problems have been solved, and document integrity is maintained at all times. No post-editing formatting is required to prepare for printing. But you cannot simply ship bit maps over the screen to get the printer to work well, if only because that would grossly underutilize the printer's ability to produce relatively high-quality output.

A variety of techniques have evolved, but the fundamental steps are the same. The PDL must generate the commands to tell the printer which fonts to use. (If it does not have fonts stored, describe the fonts by sending bit maps or mathematical descriptions of them.)

The PDL then generates the commands to tell the printer where to place the characters. It describes the graphics and takes care of bookkeeping chores, such as sending an end-of-page command when appropriate.

The first step is for the host to translate the document into pages that can then be described by the PDL. (In our generalized EPS, pagination is done in real time; some older-generation systems require a separate pagination process after editing is complete.) If the system designers want to be able to deal with several different sorts of printers and PDLs, they might decide to have the host generate an intermediate language sufficiently generic to be intelligible to various PDLs. The intermediate language might also be optimized for graphics. For example, a circle can be described in various ways, including treating it as a polygonal shape, using coefficients, or giving its center and radius. The intermediate language might optimize for data compression by describing the circle in the way that requires the least data, while still providing sufficient data for various PDLs to reconstitute it accurately.

It is perfectly possible, and not uncommon, to use no intermediate language at all. But in a network environment it is useful to separate the different phases of the translation from document to PDL, and consequently intermediate languages become more important. For example, if the document's host emits an intermediate description of the page, the printer server can translate it into the specific PDLs of the various printers it might service. Thus, one server can drive several different sorts of printers. Also, the intermediate language can be optimized for data compression, which is especially important when operating in a network environment.

Page-Description Languages

At the moment, several PDLs are vying to become the standard, although there is even debate about whether a standard is desirable. The two best-known ones are PostScript from Adobe Systems and Interpress from Xerox Corporation. A more recent contender in the standards fray is Imagen's Document Description Language (DDL).

PostScript seems to be gaining ground in the low-end desktop-publishing market. For example, PostScript drives the Apple LaserWriter. Interpress might prove to be more adept at dealing with very high speed printers such as the 120-page-per-minute Xerox 9700. DDL is a strong contender, especially since Hewlett-Packard has announced that it will support this language in some future products. One such product is a low-cost RIP that can be added to IBM PC AT systems to turn a previously installed LaserJet into a full-functionality text and graphics printer. Finally, RIPrint is the PDL we designed; we will discuss it later (see reference 1).

Interpress

Interpress was created in 1982 by Robert Sproull and William Newman at the Xerox Palo Alto Research Center. It remained for internal use only until 1984, when some versions were made public.

Interpress is a tokenized **language,** within which documents are represented in an encoded form. Each variable is preceded by a length field so that the interpreter knows in advance how much data the current token requires to be read from the input stream. This facilitates the transmission of documents quickly and reliably over a network without recomposition. This is an advantage for demand publishing on high-throughput printers, but for most users the benefit seems small.

The body of an Interpress document contains the characters to be printed and printing instructions as to the desired appearance of the document (e.g., whether

continued

DDL can produce both a user-readable and a tightly encoded form.

to print on two sides, and how to stack it in the printer). The document creator can determine and alter the coordinate system at any time, affording a measure of device independence. Generally, Interpress documents do not carry with them bit maps of characters. Bit-map images are supported for printers with resolution of about 300 dpi (the standard for low-end laser printers). Publicly available Interpress printers support limited transforms on characters, permitting, for example, rotations of only 90 degrees.

PostScript

Adobe was formed in late 1982 by John Warnock and Charles Geschke, both of whom left Xerox to design PostScript. The first PostScript-based laser printer, the Apple LaserWriter, became available in the spring of 1985.

PostScript is an interpreted language for describing, in a device-independent fashion, the way in which pages can be composed of characters, shapes, and digitized images in black and white, gray scale, or color. PostScript characters are produced by specifying a font name and size. As with Interpress, the creator can determine and alter the coordinate system at any time. Because PostScript is a rather rich programming language, a PostScript program is likely to specify graphics objects in some normalized, parameterized form and apply an appropriate transform, derived from the parameters, to the object before committing its image to the page. It is completely user-readable, in 7-bit ASCII form. For this this intelligibility, however, you pay a performance price in lower communications throughput.

When dealing with complex pages (a function of the amount of detail in the page), PostScript is severely limited. For every change in font or size, the printer must compute a bit map from the outline font (unless that font is cached). The bit map is preserved (so it does not have to be recomputed) until memory runs short; then it is discarded to make room for new bit maps. The result is that complex pages that contain text print slowly. The same is true, but more so, for pages that contain complex graphics. For example, *The Seybold Report on Publishing Systems* (the major independent industry source) reported that a map of the United States took 13 hours to typeset using PostScript. Further, the maker of the typesetter defended its performance by saying that this was because the user had encoded the graphic poorly. When the typesetter company recoded it, it still took six hours to output (see reference 2).

Document Description Language

The designers of DDL sought to avoid what they regarded as the bad design features of Interpress and PostScript, while incorporating the good. For example, DDL can produce both a user-readable and a tightly encoded form. The first allows more rapid application development, but the second allows faster host-to-printer communication.

Also, DDL has the graphics richness of PostScript, but produces document-description (as opposed to purely page-description) facilities, as does Interpress. This latter facility is important for making intelligent spooling systems that might need to know about page size, fonts needed for the entire document, where the page data ends, and so on. This kind of capability is so useful that Adobe Systems promulgated a PostScript "comment convention" by which a PostScript creator can indicate these parameters in a standard way. In effect, this convention is a recognition of PostScript's inability to deal with certain problems that arise when printing a document.

Digits to Dots: RIPs

Now that the PDL has transmitted the information describing the page, the RIP must decode the information and translate it into instructions for the printer.

The basics of a RIP are fairly uniform. In the simplest case, such as with the original Apple LaserWriter RIP, the RIP takes input, uses instructions in ROM to compose a page bit map in RAM, and then outputs it line by line through a video output section.

Let's look at this a little more closely (see figure 4). The input section receives a page description from a host computer. As the page description comes in, the 68000 CPU interprets the PDL, using the interpretation code in ROM, and generates a page bit map in RAM. When the bit map is ready, the 68000 reads a word and sends it to a shift register in the video section. The video section sends the contents of the register to the printer as a raster, much like a television signal, over a single line. The contents of the register control the laser, which in turn places the dots on the paper, producing the end result you see.

The entire operation is under the con-

continued

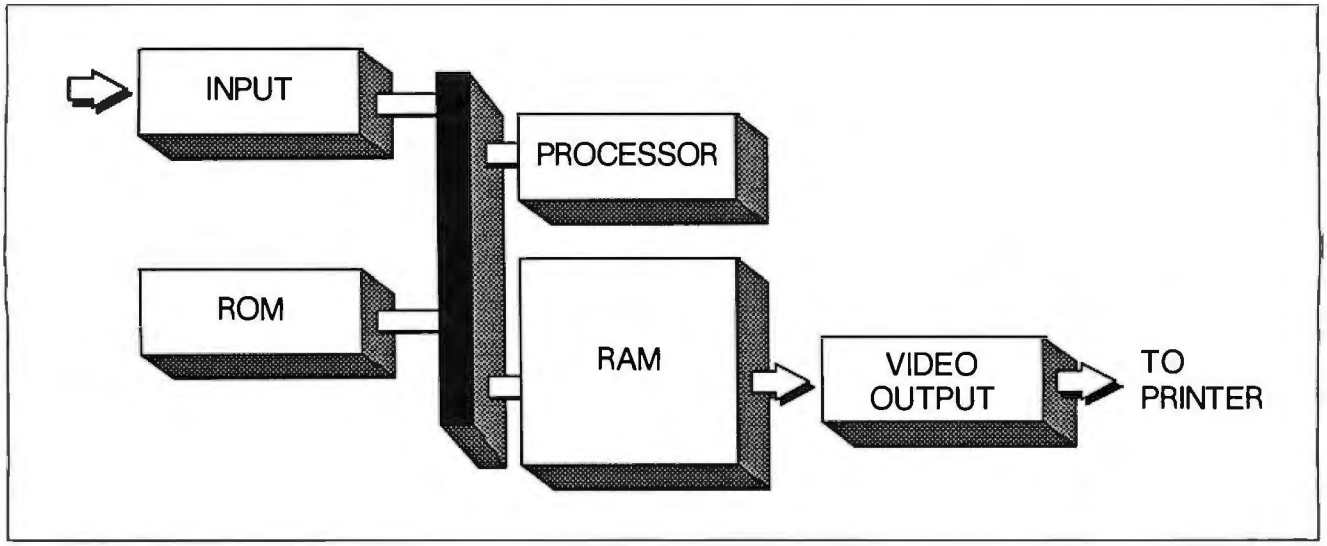

Figure 4: *A simple raster-image processor.*

A RIP needs memory for three purposes: to store code, to store glyphs, and to act as a page buffer.

trol of a 68000 CPU. In the case of Apple, data is sent either serially or through AppleTalk. The CPU handles all handshaking and timing details for inputting from the host, composing the bit map according to the ROM-based specifications, and outputting through the video section to the printer itself.

RIP Innovations

So far, we've described a generalized picture of how RIPs work. Getting more specific, we'll discuss the Interleaf RIP in particular and look at some hardware modifications that improve performance. By describing them, we can present the next level of detail in RIP technology.

In our design, we decided to allocate the RIP's memory dynamically. A RIP needs memory for three basic purposes: to store code, to store glyphs, and to act as a page buffer. While it is tempting from the hardware point of view to segregate memories (reflecting the analytic, segmented nature of most hardware design), we had reasons for leaving the memory unsegmented.

First, having separate memories for each function can require complicated memory circuits. Also, with a minimum of 2 megabytes of dynamic RAM, not only can a second 8- by 11-inch page be composed while the first is printing, but with dynamic memory allocation, the same RIP can also print 11- by 17-inch pages by using the second megabyte for the oversize page's composition. Similarly, if you need a huge amount of font storage for some task, the memory is there to be allocated.

The bit map is made by a series of reads and writes. The reading and writing occupy much of the RIP's processing time required by the 68000 CPU. Therefore, we decided to add a hardware assist in the form of a blitter.

The 68000 can address up to 16 megabytes of memory. The Interleaf RIP has 3 megabytes installed so that it can compose the next page while the current page is being printed. Because this leaves the upper 2 address bits unused, we can use these bits to select the memory's mode. The most significant bit of the address is used to select the blit mode. When the hardware sees this bit set, it knows to address the existing 3 megabytes, but in the blit mode. The data is shifted to the right, inverted, and then put on the write-enable pins of 16 dynamic RAMs. This process is then repeated for the second word, except with a shift to the left and with the memory address incremented (see figure 5). Both writes are done in one write instruction. The data input pins on the RAMs during this process may all be set to ones or zeros. If set to one, the result is an OR. If zero, the result is a KO, or knock out (i.e., every one in the source data results in a zero in RAM).

This is a direct OR to memory, done in one cycle. It speeds up the drawing of glyphs and bit-map images by a factor of eight. A character of up to 32 by 32 pixels that are not word-aligned, for example, may be ORed to the bit map in less than 200 microseconds.

The word-boundary offset (n) arrives via its own special hardware path. The 68000 cannot address particular bits within words in one cycle, which is precisely what we need it to do. So we created a special bit-address register that contains, in 4 bits, the information necessary to address a bit within a word.

The system also has direct-memory addressing for the output to the printer, for the input from the host, and for refreshing the dynamic RAMs used. The DMA output to the printer can also be used to clear the bit map, saving approximately 0.25 second of processor time per page. To enable this function, the DMA controller uses an otherwise undefined function code when it reads the data from the

continued

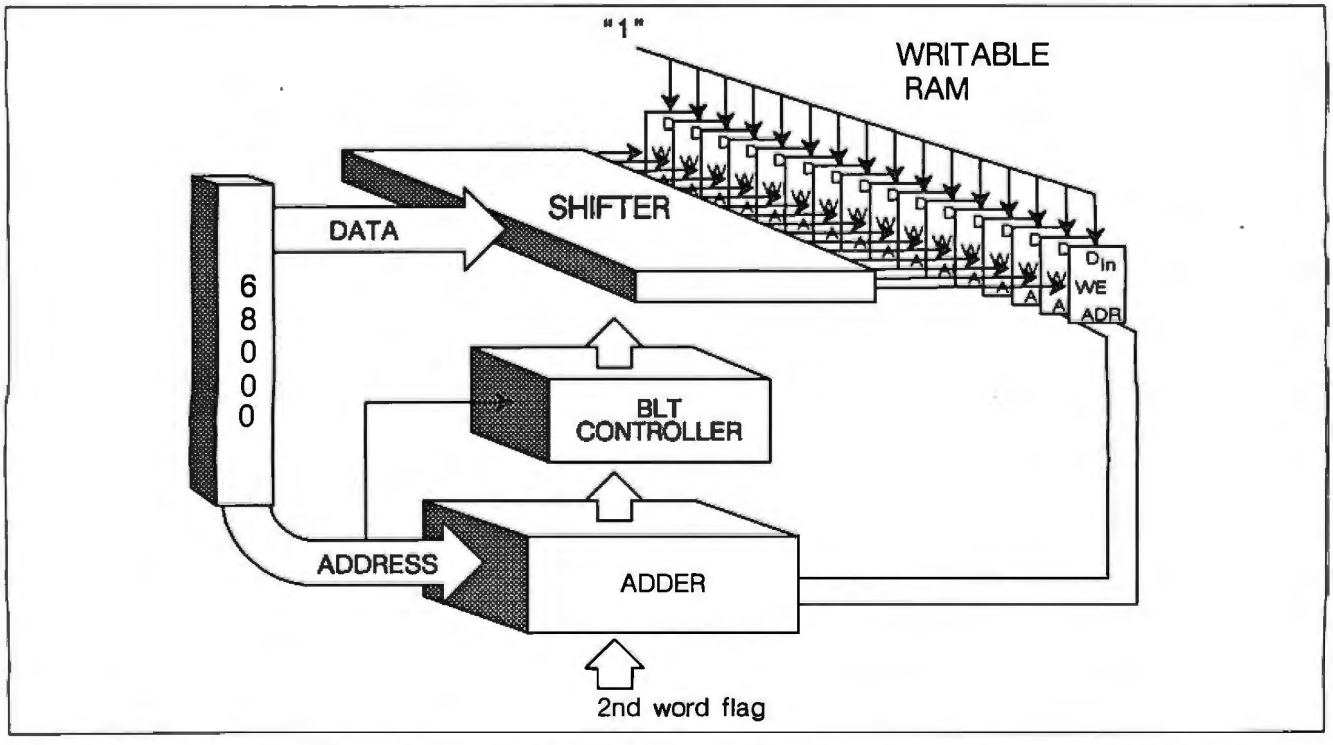

Figure 5: *The most significant flags blit mode. Data is shifted and sent to the write-enable pin of the dynamic RAMs. For the second word, the address is incremented. The result is a direct OR to memory in one cycle.*

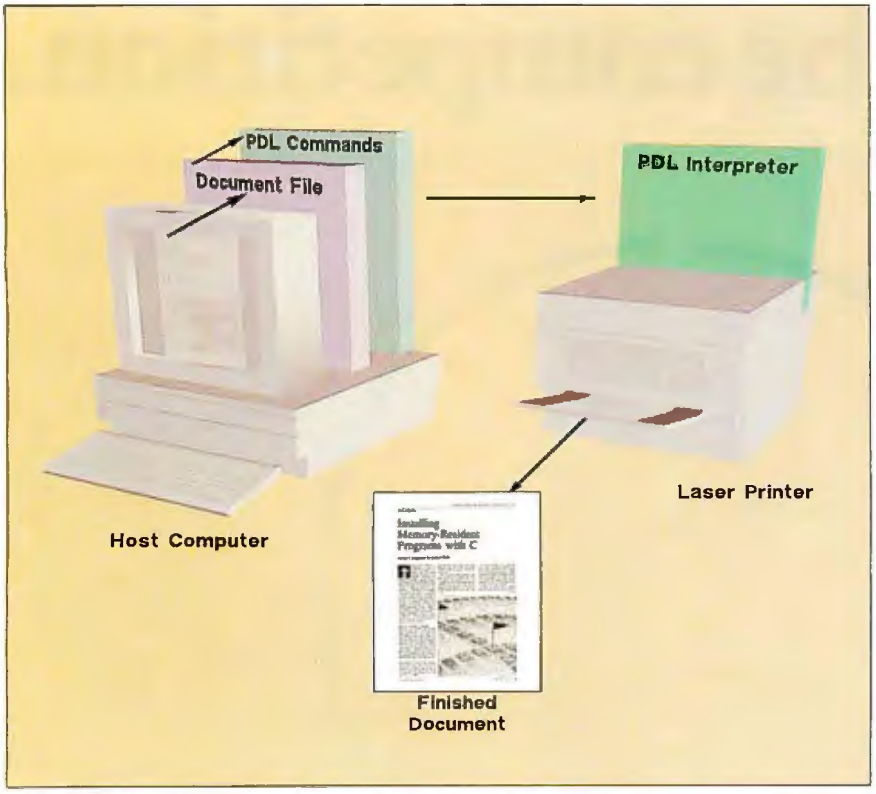

Figure 6: *The document is created on the host computer and converted into a document file, in this case a Printerleaf file. A filter translates that file into the appropriate PDL. The PDL commands are passed to the PDL interpreter on the RIP. The RIP translates the commands in the pulses of the laser, which produces the finished document.*

memory for the printer. This function code causes the memory controller to do a read-clear cycle rather than a read cycle. If you want multiple copies of a page, the bit map is not cleared.

These changes in design make a RIP capable of printing pages of great complexity at very high speed.

Software Decisions

Interleaf also designed its own document description language (Printerleaf), comparable to Interpress in many of its design considerations, and its own PDL (RIPrint) to take special advantage of the design of our RIP. The design considerations are, we think, illustrative of the challenges PDLs create.

We knew that the output from our electronic publishing software would be directed toward a wide variety of printing devices, ranging from laser printers to typesetters. Thus, we decided that the most flexible approach was to provide printing systems capable of supporting device and language independence so that documents could be transparently directed toward many existing and to-be-announced printers. Also, we recognized that many users would want to direct output to a choice of printers in an office setting that might contain various printers roughly in the same class (e.g., 300-dpi printers). In such a situation, you do not need device independence with the performance penalty it inevitably exacts (e.g., passing 2000 dpi's worth of information to a laser printer capable of only 300 dpi).

Further, we decided our PDL would be resolution-dependent, although the code would be device-independent. By making it resolution-dependent, we hoped to gain speed, quality, and control. We would gain speed because we would not have to go through the complicated algorithms necessary to translate a generic, resolution-independent, mathematical description of a font into bit maps; we would download the bit maps directly and save transmission time for lower resolution printers. In short, resolution dependence means more work at our end but produces qualities users desire.

The overall RIP software design allows for any selection of PDL interpreters to be either resident on, or downloaded to, the RIP. The RIP software is divided into the low-level RIP system interface and the higher-level PDL interpreters. The lower-level software provides communications services, print-engine interface services, memory management services, and access to functions such as bit( ), which might have hardware support. This layer of software shields the PDL interpreters from any particular hardware implementation dependencies. It also modularizes the software, easing ports to new RIP hardware implementations and adaptation to new print engines.

Our PDL, RIPrint, includes an extensive set of commands for drawing on the page, as well as controls for the printer. A RIPrint file is actually a series of immediately executed commands. Because they are immediately executed, not buffered, there is no limit to the number of commands you can use to describe a page.

The commands are encoded in binary, not ASCII. This results in compact code and efficient machine-to-machine transfer; the time savings can be orders of magnitude. If the most significant byte is set, the byte is considered a high-level command. If the MSB is not set, it is considered the Draw Thyself command. In fact, a single byte can cause a character to draw itself and set the XY position of the next character. The four Boolean operators included (OR, KO, MOV, and XOR) can, unlike with most typesetters, be applied to both text and graphics.

Summary

In general, the method for getting a document on the screen to paper is as follows (see figure 6): The document editor on the host computer produces the document file. Then a filter translates the document file into the PDL for the target printer. The PDL commands are then passed as a data stream to the PDL interpreter on board the RIP. The RIP translates the commands into the pulses of the laser in the laser printer. These pulses cause the toner to adhere to the paper to produce the printed words and graphics.

The result is a successful interaction of hardware and software to produce a high-quality document with minimum effort on the part of the user. ■

ACKNOWLEDGMENTS

We would like to thank Michael Mark, Riaz Moledina, Robert Morris, and David Weinberger for their help.

REFERENCES

1. We wish to thank Robert A. Morris, of Interleaf and the University of Massachusetts at Boston, for his contribution to this section on PDLs, much of which is drawn from his article, "Page Description Languages," Proceedings of the ProText II Conference, Boole Press, Dublin, 1985.

2. "PostScript: Can It Cut the Mustard?" *The Seybold Report on Publishing Systems*, March 17, 1986, Vol. 15, No. 12, page 20.