**Software**

---

**T\TeX\textregistered 2 MEMORY STRUCTURE**

Editor's note: The following diagram shows the relative amount of memory required by \TeX\textregistered 2 for various tables and work areas. It has been redrawn (not using \TeX, although that would be a good problem) from a slide that Don Knuth prepared for the lecture on data structures at the \TeX\textregistered 2 Short Course held in conjunction with the July TUG meeting.

Memory represented in this diagram is approximately 280K bytes, plus (not shown) 100K of dynamic memory. On a DEC 20, \TeX\textregistered 2 occupies total memory of between 350K and 600K bytes, excluding Pascal run-time requirements.

---

Editor's note: David Fuchs has provided the following copy describing the format of \texttt{.dvi} files, including some changes made since the July TUG meeting. A summary of the changes appears in David's "News from Stanford" article, which begins on page 20.
1. Device-independent file format. The most important output produced by a run of \TeX{} is the “device independent” (DVI) file that specifies where characters and rules are to appear on printed pages. The form of these files was designed by David R. Fuchs in 1979. Almost any reasonable device can be driven by a program that takes DVI files as input, and dozens of such DVI-to-whatever programs have been written. Thus, it is possible to print the output of \TeX{} on many different kinds of equipment, using \TeX{} as a device-independent “front end.”

A DVI file is a stream of 8-bit bytes, which may be regarded as a series of commands in a machine-like language. The first byte of each command is the operation code, and this code is followed by zero or more bytes that provide parameters to the command. The parameters themselves may consist of several consecutive bytes; for example, the ‘set_rule’ command has two parameters, each of which is four bytes long. Parameters are usually regarded as nonnegative integers; but four-byte-long parameters, and shorter parameters that denote distances, can be either positive or negative. Such parameters are given in two’s complement notation. For example, a two-byte-long distance parameter has a value between $-2^{15}$ and $2^{15} - 1$.

A DVI file consists of a “preamble,” followed by a sequence of one or more “pages,” followed by a “postamble.” The preamble is simply a pre command, with its parameters that define the dimensions used in the file; this must come first. Each “page” consists of a bop command, followed by any number of other commands that tell where characters are to be placed on a physical page, followed by an eop command. The pages appear in the order that \TeX{} generated them. If we ignore nap commands and fmt.def commands (which are allowed between any two commands in the file), each eop command is immediately followed by a bop command, or by a post command; in the latter case, there are no more pages in the file, and the remaining bytes form the postamble. Further details about the postamble will be explained later.

Some parameters in DVI commands are “pointers.” These are four-byte quantities that give the location number of some other byte in the file; the first byte is number 0, then comes number 1, and so on. For example, one of the parameters of a bop command points to the previous bop; this makes it feasible to read the pages in backwards order, in case the results are being directed to a device that stacks its output face up. Suppose the preamble of a DVI file occupies bytes 0 to 99. Now if the first page occupies bytes 100 to 999, say, and if the second page occupies bytes 1000 to 1999, then the bop that starts in byte 1000 points to 100 and the bop that starts in byte 2000 points to 1000. (The very first bop, i.e., the one that starts in byte 100, has a pointer of $-1$.)

2. The DVI format is intended to be both compact and easily interpreted by a machine. Compactness is achieved by making most of the information implicit instead of explicit; when a DVI-reading program reads the commands for a page, it keeps track of several quantities: (a) The current font \textit{f} is an integer; this value is changed only by fmt and fmt.num commands. (b) The current position on the page is given by two numbers called the horizontal and vertical coordinates, \textit{h} and \textit{v}. Both coordinates are zero at the upper left corner of the page; moving to the right corresponds to increasing the horizontal coordinate, and moving down corresponds to increasing the vertical coordinate. Thus, the coordinates are essentially Cartesian, except that vertical directions are flipped; the Cartesian version of \textit{(h,v)} would be \textit{(h, -v)}. (c) The current spacing amounts are given by four numbers \textit{w}, \textit{x}, \textit{y}, and \textit{z}, where \textit{w} and \textit{z} are used for horizontal spacing and where \textit{y} and \textit{z} are used for vertical spacing. (d) There is a stack containing \textit{(h,v,w,x,y,z)} values; the DVI commands push and pop are used to change the current level of operation. Note that the current font \textit{f} is not pushed and popped; the stack contains only information about positioning.

The values of \textit{h}, \textit{v}, \textit{w}, \textit{x}, \textit{y}, and \textit{z} are signed integers having up to 32 bits; including the sign. Since they represent physical distances, there is a small unit of measurement such that increasing \textit{h} by 1 means moving a certain tiny distance to the right. The actual unit of measurement is variable, as explained below; \TeX{} sets things up so that its DVI output is in sp units, i.e., scaled points, in agreement with all the scaled dimensions in \TeX{}’s data structures.
8. Here is list of all the commands that may appear in a DVI file. With each command we give its symbolic name (e.g., \texttt{bop}), its opcode byte (e.g., 129), and its parameters (if any). The parameters are followed by a bracketed number telling how many bytes they occupy; for example, \texttt{p[4]} means that parameter \textit{p} is four bytes long.

\texttt{set.char.0} 0. Typeset character number 0 from font \textit{f} such that the reference point of the character is at \((h,v)\). Then increase \textit{h} by the width of that character. Note that a character may have zero or negative width, so one cannot be sure that \textit{h} will advance after this command; but \textit{h} usually does increase.

\texttt{set.char.1} through \texttt{set.char.127} (opcodes 1 to 127). Do the operations of \texttt{set.char.0}, but use the appropriate character number instead of character 0.

\texttt{setl} 128 \textit{c[1]}. Same as \texttt{set.char.0}, except that character number \textit{c} is typeset. \TeX82 uses this command for characters in the range \(128 \leq c < 256\).

\texttt{setq} 129 \textit{c[2]}. Same as \texttt{setl}, except that \textit{c} is two bytes long, so it is in the range \(0 \leq c < 65536\). \TeX82 never uses this command, but it should come in handy for extensions of \TeX that deal with oriental languages.

\texttt{set3} 130 \textit{c[3]}. Same as \texttt{setl}, except that \textit{c} is three bytes long, so it can be as large as \(2^{24} - 1\). Not even the Chinese language has this many characters, but this command might prove useful in some yet unforeseen extension.

\texttt{set4} 131 \textit{c[4]}. Same as \texttt{setl}, except that \textit{c} is four bytes long. Imagine that.

\texttt{set.rule} 132 \textit{a[4]} \textit{b[4]}. Typeset a solid black rectangle of height \textit{a} and width \textit{b}, with its bottom left corner at \((h,v)\). Then set \textit{h} \leftarrow \textit{h} + \textit{b}. If either \textit{a} \leq 0 or \textit{b} \leq 0, nothing should be typeset. Note that if \textit{b} < 0, the value of \textit{h} will decrease even though nothing else happens. See below for details about how to typeset rules so that consistency with \texttt{METAFONT} is guaranteed.

\texttt{put} 133 \textit{c[1]}. Typeset character number \textit{c} from font \textit{f} such that the reference point of the character is at \((h,v)\). (The 'put' commands are exactly like the 'set' commands, except that they simply put out a character or a rule without moving the reference point afterwards.)

\texttt{put2} 134 \textit{c[2]}. Same as \texttt{set2}, except that \textit{h} is not changed.

\texttt{put3} 135 \textit{c[3]}. Same as \texttt{set3}, except that \textit{h} is not changed.

\texttt{put4} 136 \textit{c[4]}. Same as \texttt{set4}, except that \textit{h} is not changed.

\texttt{put.rule} 137 \textit{a[4]} \textit{b[4]}. Same as \texttt{set.rule}, except that \textit{h} is not changed.

\texttt{nop} 138. No operation, do nothing. Any number of \texttt{nop}'s may occur between DVI commands, but a \texttt{nop} cannot be inserted between a command and its parameters or between two parameters.

\texttt{bop} 139 \textit{c0[4]} \textit{c1[4]} \ldots \textit{cn[4]} \textit{p[4]}. Beginning of a page: Set \((h,v,w,x,y,z) \leftarrow (0,0,0,0,0,0)\) and set the stack empty. Set the current font \textit{f} to an undefined value. The ten \textit{c\_i} parameters hold the values of \texttt{\_count\_0} \ldots \texttt{\_count\_9} in \TeX at the time \texttt{\_shipout} was invoked for this page; they can be used to identify pages, if a user wants to print only part of a DVI file. The parameter \textit{p} points to the previous \texttt{bop} command in the file, where the first \texttt{bop} has \textit{p} = \texttt{\_shipout}.

\texttt{cop} 140. End of page: Print what you have read since the previous \texttt{bop}. At this point the stack should be empty. (The DVI-reading programs that drive most output devices will have kept a buffer of the material that appears on the page that has just ended. This material is largely, but not entirely, in order by \textit{v} coordinate and (for fixed \textit{v}) by \textit{h} coordinate; so it usually needs to be sorted into some order that is appropriate for the device in question.)

\texttt{push} 141. Push the current values of \((h,v,w,x,y,z)\) onto the top of the stack; do not change any of these values. Note that \textit{f} is not pushed.

\texttt{pop} 142. Pop the top six values off the stack and assign them respectively to \((h,v,w,x,y,z)\). The number of pops should never exceed the number of pushes, since it would be highly embarrassing if the stack were empty at the time of a \texttt{pop} command.

\texttt{right} 143 \textit{b[1]}. Set \textit{h} \leftarrow \textit{h} + \textit{b}, i.e., move right \textit{b} units. The parameter is a signed number in two's complement notation, \(-128 \leq b < 128\); if \textit{b} < 0, the reference point actually moves left.
right2 144 b[2]. Same as right1, except that b is a two-byte quantity in the range $-32768 \leq b < 32768$.

right3 145 b[3]. Same as right1, except that b is a three-byte quantity in the range $-2^{23} \leq b < 2^{23}$.

right4 146 b[4]. Same as right1, except that b is a four-byte quantity in the range $-2^{31} \leq b < 2^{31}$.

w0 147. Set $h \leftarrow h + w$; i.e., move right w units. With luck, this parameterless command will usually suffice, because the same kind of motion will occur several times in succession; the following commands explain how w gets particular values.

w1 148 b[1]. Set $w \leftarrow b$ and $h \leftarrow h + b$. The value of b is a signed quantity in two's complement notation, $-128 \leq b < 128$. This command changes the current w spacing and moves right by b.

w2 149 b[2]. Same as w1, but b is two bytes long, $-32768 \leq b < 32768$.

w3 150 b[3]. Same as w1, but b is three bytes long, $-2^{23} \leq b < 2^{23}$.

w4 151 b[4]. Same as w1, but b is four bytes long, $-2^{31} \leq b < 2^{31}$.

x0 152. Set $h \leftarrow h + x$; i.e., move right x units. The 'z' commands are like the 'w' commands except that they involve x instead of w.

z1 153 b[1]. Set $x \leftarrow b$ and $h \leftarrow h + b$. The value of b is a signed quantity in two's complement notation, $-128 \leq b < 128$. This command changes the current x spacing and moves right by b.

z2 154 b[2]. Same as z1, but b is two bytes long, $-32768 \leq b < 32768$.

z3 155 b[3]. Same as z1, but b is three bytes long, $-2^{23} \leq b < 2^{23}$.

z4 156 b[4]. Same as z1, but b is four bytes long, $-2^{31} \leq b < 2^{31}$.

downl 157 a[1]. Set $v \leftarrow v + a$, i.e., move down a units. The parameter is a signed number in two's complement notation, $-128 \leq a < 128$; if $a < 0$, the reference point actually moves up.

down2 158 a[2]. Same as down1, except that a is a two-byte quantity in the range $-32768 \leq a < 32768$.

down3 159 a[3]. Same as down1, except that a is a three-byte quantity in the range $-2^{23} \leq a < 2^{23}$.

down4 160 a[4]. Same as down1, except that a is a four-byte quantity in the range $-2^{31} \leq a < 2^{31}$.

y0 161. Set $v \leftarrow v + y$; i.e., move down y units. With luck, this parameterless command will usually suffice, because the same kind of motion will occur several times in succession; the following commands explain how y gets particular values.

y1 162 a[1]. Set $y \leftarrow a$ and $v \leftarrow v + a$. The value of a is a signed quantity in two's complement notation, $-128 \leq a < 128$. This command changes the current y spacing and moves down by a.

y2 163 a[2]. Same as y1, but a is two bytes long, $-32768 \leq a < 32768$.

y3 164 a[3]. Same as y1, but a is three bytes long, $-2^{23} \leq a < 2^{23}$.

y4 165 a[4]. Same as y1, but a is four bytes long, $-2^{31} \leq a < 2^{31}$.

x0 166. Set $v \leftarrow v + z$; i.e., move down z units. The 'z' commands are like the 'y' commands except that they involve z instead of y.

z1 167 a[1]. Set $z \leftarrow a$ and $v \leftarrow v + a$. The value of a is a signed quantity in two's complement notation, $-128 \leq a < 128$. This command changes the current z spacing and moves down by a.

z2 168 a[2]. Same as z1, but a is two bytes long, $-32768 \leq a < 32768$.

z3 169 a[3]. Same as z1, but a is three bytes long, $-2^{23} \leq a < 2^{23}$.

z4 170 a[4]. Same as z1, but a is four bytes long, $-2^{31} \leq a < 2^{31}$.

fnt.num.0 171. Set $f \leftarrow 0$. Font 0 must previously have been defined by a fnt.def instruction, as explained below.

fnt.num.1 through fnt.num.63 (opcodes 172 to 234). Set $f \leftarrow 1, \ldots, f \leftarrow 63$, respectively.

fnt 235 k[1]. Set $f \leftarrow k$. TEx82 uses this command for font numbers in the range $64 \leq k < 256$.

fnt2 236 k[2]. Same as fnt1, except that k is two bytes long, so it is in the range $0 \leq k < 65536$. TEx82 never generates this command, but large font numbers may prove useful for specifications of color or texture, or they may be used for special fonts that have fixed numbers in some external coding scheme.
\texttt{fmt3} 237 \texttt{k[3]}. Same as \texttt{fmt1}, except that \texttt{k} is three bytes long, so it can be as large as $2^{24} - 1$.

\texttt{fmt4} 238 \texttt{k[4]}. Same as \texttt{fmt1}, except that \texttt{k} is four bytes long; this is for the really big font numbers (and for the negative ones).

\texttt{zzz1} 239 \texttt{k[1] z[k]}. This command is undefined in general; it functions as a \texttt{(k + 2)-byte nop} unless special DVI-reading programs are being used. \TeX{} generates \texttt{zzz1} when a normal \texttt{\textbackslash send} appears, setting \texttt{k} to the number of bytes being sent. It is recommended that \texttt{z} be a string having the form of a keyword followed by possible parameters relevant to that keyword.

\texttt{zzz2} 240 \texttt{k[2] z[k]}. Like \texttt{zzz1}, but \texttt{0 \leq k < 65536}.

\texttt{zzz3} 241 \texttt{k[3] z[k]}. Like \texttt{zzz1}, but \texttt{0 \leq k < 2^{24}}.

\texttt{zzz4} 242 \texttt{k[4] z[k]}. Like \texttt{zzz1}, but \texttt{k} can be ridiculously large. \TeX{} uses \texttt{zzz4} when sending a string of length 256 or more.


\texttt{post} 248. Beginning of the postamble, see below.

\texttt{post\_post} 249. Ending of the postamble, see below.

Commands 250–255 are undefined at the present time.

\textbf{4.} The preamble contains basic information about the file as a whole. As stated above, there are six parameters:


The \texttt{i} byte identifies DVI format; currently this byte is always set to 2. (Some day we will set \texttt{i = 3,} when DVI format makes another incompatible change—perhaps in 1992.)

The next two parameters, \texttt{num} and \texttt{den}, are positive integers that define the units of measurement; they are the numerator and denominator of a fraction by which all dimensions in the DVI file could be multiplied in order to get lengths in units of 10^{-7} meters. Since there are 72.272 points per inch and 2.54 centimeters per inch, and since \TeX{} works with scaled points where there are $2^{16}$ sp in a point, \TeX{} sets \texttt{num = 25400000} and \texttt{den = 7227 \cdot 2^{16} = 473628672}.

The \texttt{mag} parameter is what \TeX{} calls \texttt{\textbackslash mag}, i.e., 1000 times the desired magnification. The actual fraction by which dimensions are multiplied is therefore \texttt{mag \cdot num/1000den}. Note that if a \TeX{} source document does not call for any ‘true’ dimensions, and if you change it only by specifying a different \texttt{\textbackslash mag} setting, the DVI file that \TeX{} creates will be completely unchanged except for the value of \texttt{mag} in the preamble and postamble. (Fancy DVI-reading programs allow users to override the \texttt{mag} setting when a DVI file is being printed.)

Finally, \texttt{k} and \texttt{z} allow the DVI writer to include a comment, which is not interpreted further. The length of comment \texttt{z} is \texttt{k}, where \texttt{0 \leq k < 256}.

\texttt{define id\_byte = 2} \{ identifies the kind of DVI files described here \}
5. Font definitions for a given font number \( k \) contain further parameters


The four-byte value \( c \) is the check sum that \( \TeX \) found in the TFM file for this font; \( c \) should match the check sum of the font found by programs that read this DVI file.

Parameter \( s \) contains a fixed-point scale factor that is applied to the character widths in font \( k \); font dimensions in TFM files and other font files are relative to this quantity, which is called the "at size" elsewhere in this documentation. The value of \( s \) is always positive and less than \( 2^{27} \). It is given in the same units as the other DVI dimensions, i.e., in sp when \( \TeX \) has made the file. Parameter \( d \) is similar to \( s \); it is the "design size," and it is given in DVI units that have not been corrected for the magnification \( \text{mag} \) found in the preamble. Thus, font \( k \) is to be used at \( \text{mag} \cdot s/1000d \) times its normal size.

The remaining part of a font definition gives the external name of the font, which is an ascii string of length \( a + l \). The number \( a \) is the length of the "area" or directory, and \( l \) is the length of the font name itself; the standard local system font arch is supposed to be used when \( a = 0 \). The \( n \) field contains the area in its first \( a \) bytes.

Font definitions must appear before the first use of a particular font number. Once font \( k \) is defined, it must not be defined again; however, we shall see below that font definitions appear in the postamble as well as in the pages, so in this sense each font number is defined exactly twice, if at all. Like \( \text{nop} \) commands, font definitions can appear before the first \( \text{bop} \), or between an \( \text{pop} \) and a \( \text{bop} \).

6. Sometimes it is desirable to make horizontal or vertical rules line up precisely with certain features in characters of a font. It is possible to guarantee the correct matching between DVI output and the characters generated by \METAFONT by adhering to the following principles:

1. The \METAFONT characters should be positioned so that a bottom edge or left edge that is supposed to line up with the bottom or left edge of a rule appears at the reference point, i.e., in row 0 and column 0 of the \METAFONT raster. This ensures that the position of the rule will not be rounded differently when the pixel size is not a perfect multiple of the units of measurement in the DVI file. (2) A typeset rule of height \( a > 0 \) and width \( b > 0 \) should be equivalent to a \METAFONT-generated character having black pixels in precisely those raster positions whose \METAFONT coordinates satisfy \( 0 \leq x < ab \) and \( 0 \leq y < aa \), where \( a \) is the number of pixels per DVI unit.

7. The last page in a DVI file is followed by ‘post’; this command introduces the postamble, which summarizes important facts that \( \TeX \) has accumulated about the file, making it possible to print subsets of the data with reasonable efficiency. The postamble has the form

\[
\langle \text{font definitions} \rangle \\
\]

Here \( p \) is a pointer to the final \( \text{bop} \) in the file. The next three parameters, \text{num}, \text{den}, and \text{mag}, are duplicates of the quantities that appeared in the preamble.

Parameters \( l \) and \( u \) give respectively the height-plus-depth of the tallest page and the width of the widest page, in the same units as other dimensions of the file. These numbers might be used by a DVI-reading program to position individual "pages" on large sheets of film or paper.

Parameter \( s \) is the maximum stack depth (i.e., the largest excess of \text{push} commands over \text{pop} commands) needed to process this file. Then comes \( t \), the total number of pages (\text{bop} commands) present.

The postamble continues with font definitions, which are any number of \text{fmt.def} commands as described above, possibly interspersed with \text{nop} commands. Each font number that is used in the DVI file must be defined exactly twice: Once before it is first selected by a \text{fmt} command, and once in the postamble.
8. The last part of the postamble, following the \texttt{post} \texttt{post} byte that signifies the end of the font definitions, contains \texttt{q}, a pointer to the \texttt{post} command that started the postamble. An identification byte, \texttt{i}, comes next; this currently equals 2, as in the preamble.

The \texttt{i} byte is followed by four or more bytes that are all equal to the decimal number 223 (i.e., '337 in octal). \TeX{} puts out four to seven of these trailing bytes, until the total length of the file is a multiple of four bytes, since this works out best on machines that pack four bytes per word; but any number of 223's is allowed, as long as there are at least four of them. In effect, 223 is a sort of signature that is added at the very end.

This curious way to finish off a \TeX{} file makes it feasible for \TeX{}-reading programs to find the postamble first, on most computers, even though \TeX{} wants to write the postamble last. Most operating systems permit random access to individual words or bytes of a file, so the \TeX{} reader can start at the end and skip backwards over the 223's until finding the identification byte. Then it can back up four bytes, read \texttt{q}, and move to byte \texttt{q} of the file. This byte should, of course, contain the value 248 (\texttt{post}); now the postamble can be read, so the \TeX{} reader discovers all the information needed for typesetting the pages. Note that it is also possible to skip through the \TeX{} file at reasonably high speed to locate a particular page, if that proves desirable. This saves a lot of time, since \TeX{} files used in production jobs tend to be large.

Unfortunately, however, standard PASCAL does not include the ability to access a random position in a file, or even to determine the length of a file. Almost all systems nowadays provide the necessary capabilities, so \TeX{} format has been designed to work most efficiently with modern operating systems. But if \TeX{} files have to be processed under the restrictions of standard PASCAL, one can simply read them from front to back, since the necessary header information is present in the preamble and in the font definitions. (The \texttt{l} and \texttt{u} and \texttt{s} and \texttt{t} parameters, which appear only in the postamble, are "frills" that are handy but not absolutely necessary.)