Digital Equipment Corporation
Maynard, Massachusetts

digital

Markup and
Perforation Guide

TYPESET-8

Photon 513/560
Display Ad Program

DEC-08-H8QA-D

PHOTON 513/560 DISPLAY AD PROGRAM

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

1st Printing October 1969

CONTENTS

CONTENTS (Cont)

CHAPTER 1
INTRODUCTION

1.1 THE PDP-8 PHOTON 513/560 SYSTEM

The PDP-8 Photon 513/560 System utilizes a computer to produce punched paper tape input to the forward-reading Photon 513 or the Photon 560. To accomplish this process, input tapes (tapes containing unjustified, unhyphenated text interspersed with controls indicating the desired format or appearance of the printed output) must be manually punched and fed into the PDP-8 computer. This manual explains the preparation of these input tapes.

Input tapes may be punched on any tape-perforating device having a 63-key keyboard and producing 6-level tape. It is assumed that the user is familiar with the operation of such devices.

The procedure for using the PDP-8 Photon 513/560 System begins with the mark-up of the copy to be printed. The following chapter of this manual describes how and where (within the text) to specify the information necessary for obtaining the desired output. This same information should appear as part of the mark-up, in the appropriate places, to insure that it will be keypunched along with the text.

The next step of the process is the punching of the input tape — tape containing unjustified, unhyphenated text interspersed with controls (punched exactly as specified in this manual).

After the input tape has been punched, the rest of the process is automatic. The input tape is placed in the PDP-8 reader. The PDP-8 begins reading the tape, while simultaneously punching another tape — the output tape. The output tape contains the text, along with controls which cause the Photon 513 or the Photon 560 to produce justified text (hyphenated and letter spaced when necessary) in the format specified by the input tape. This reading and punching of tape by the PDP-8 is continuous until the entire input tape has been read.

The final step of the process consists of taking the output tape from the PDP-8 and placing it in the Photon 513 or Photon 560 reader. The Photon 513 or Photon 560 then begins reading the tape and producing the printed text as specified by the controls punched by the PDP-8 on the output tape.

The key to the operation of the system lies in properly specifying the controls on the input tape. The following chapter describes all the controls which are available in the system, and how each is used. Chapter 3 discusses a feature of the system which can greatly reduce the amount of work involved in specifying controls — the setting up of standard formats to be used for different pieces of copy having different text, but the same format.

Frequently, the user will find that there is more than one way to specify a desired output. Often, the choice is one of personal taste. By efficient use of the controls, however, the amount of time, re-punching, and paste-up involved in setting copy can be kept to a minimum.

CHAPTER 2
FUNCTIONS

2.1 GENERAL

This chapter describes the functions which can be performed by the PDP-8 Photon 513/560 System; that is, the information which must be keypunched along with the text in order to achieve the desired printed output. Some functions can be achieved by punching a single key, such as quad left or insert leaders. Others must be described by a command. Commands are punched in the following format: A command begins with a $ followed by two letters. The first letter tells what group of commands the command belongs to, the second letter tells what function the command performs. These two letters may be followed by one or more numbers, depending upon the command. At the end of the command, a return is punched.

The specific format for each command is discussed in the following pages. In the description, lower case letters will frequently be used to represent numbers. For example, $CPx,y(RET) is a command beginning with a $, followed by the two letters CP. These are then followed by two numbers (which we are calling x and y) separated by a comma. A return is then punched to indicate that this is the end of the command. It is important that the format for each command be followed exactly as given in this manual; no blank spaces or extra punctuation can be added within a command. If the proper format is not used, an error will result and the command will not be executed. (See Appendix III) It is advisable to punch some tape-feed before and after all commands so that the commands can be corrected, if necessary, without repunching the entire input tape.

Three important rules of format are the following:

a. A $ is always used to mean the beginning of a command. If a $ is to be printed out as part of the text, two dollar signs must be punched, ($$). This will result in one $ being printed.

b. In the description of the commands in this handbook, the two letters following the $ are typed in capital letters. However, when punching the input tape, these letters may be punched in either upper or lower case, but the $ must be punched in lower case.

c. If two or more commands are punched one after the other, each command must be ended by a return before the next command is punched. For example:

```
$CW12(RET)$CT3(RET)
```

The first (RET) cannot be omitted.

2.2 ESSENTIAL PARAMETER COMMANDS

Type face, point size, column width, set size and leading are selected by commands in the C-group; that is, these commands all begin with $C followed by another letter. It is important that the

type face
point size
set size
column width
leading

be specified at the beginning of every input tape, before any text is punched; otherwise, none of the text will be processed. It is suggested that a considerable number of tape feeds be left at the beginning of each tape, so that any missing parameters can be added to the tape (instead of repunching the entire tape).

These five parameters may be specified by the individual commands described below, or by the $CC command (see Section 2.2.5) which specifies all of them simultaneously. Any or all of these five parameters may be changed at any point within a take by use of the C-group commands.

2.2.1 Selecting Type Face (Style)

The type face to be used is set by punching the command $CTx(RET) where x is a number from 1 to 16, corresponding to the type face desired. Example: $CT4(RET) would call for type face 4 on the Photon disc.

All text following a $CT command will be set in the specified type face until the next $CT or $CC command is punched, changing the type face. The change is effective immediately, thus allowing the mixing of any number of type faces within the same line.

A type face must be selected at the beginning of each input tape, before any text is punched.

2.2.2 Selecting Point Size (Lens) and Set Size

Point size and set size are selected by punching the command $CPx,y(RET) where x is the point size and y is the set size; a comma is punched between x and y. If the set size is omitted, the system will calculate the appropriate set size, depending upon the type face and point size specified. If the second number is omitted, the comma should also be omitted and the command $CPx(RET) used. If the point size (or set size) specified is not one which is available on the Photon, the closest available smaller size is substituted for it.

Examples:

$CP12,12.5(RET) would call for point size 12, set size 12-1/2.
$CP18(RET) would call for point size 18 and appropriate set size.
$CP20(RET) would call for point size 24 (the nearest smaller one) and appropriate set size.

All text following a $CP command will be set in the specified point size and set size, until the next $CP or $CC command is punched. The new point size and set size specified will be effective immediately, thus allowing the mixing of point sizes within the same line.

A point size must be specified at the beginning of each input tape, before any text is punched.

2.2.3 Setting Column Width (Line Length)

Column width is set by punching the command $CWx(RET) where x is the width in picas and points. The picas and points are separated by a period. If the number of points is zero, the period and the points may be omitted. The column width must be less than 54 picas.

Examples:

> $CW14.9(RET) will set column width to 14 picas, 9 points.
> $CW11.0(RET) will set column width to 11 picas.
> $CW11(RET) will set column width to 11 picas.

This command can be used to change the column width at any time during a take. If the command is punched in the middle of a line of text, the new width will not be effective until after the current line is photographed.

A column width must be specified at the beginning of each input tape, before any text is punched.

2.2.4 Setting Leading

Stored leading is set by punching the command $CLx(RET) where x is a number from 0 to 63 indicating the number of points of stored leading desired. If a number greater than 63 is given, the stored leading will be set to 63 points.

Example:

> $CL24(RET) would cause the film to be advanced 24 points after a line was photographed.

All lines following a $CL command will be set with the specified leading, until another stored leading command is punched. If a $CL command is punched in the middle of a line of text, the new leading will be effective at the end of the current line of text. The last stored leading command encountered before the line is photographed will be the one used.

The amount of stored leading must be specified at the beginning of each input tape, before any text is punched.

Leading may be added anywhere in a line by punching the command $CAx(RET) where x is a number from 0 to 255, indicating the number of points of leading to be added.

Example:

ABC$CA12(RET)DEF would be printed as

```
ABC
   DEF
```

Using a $CA command does not affect the stored leading. The $CA command causes a film advance at that position of the line at which it is encountered. At the end of that line, the film advance is determined by the stored leading (which may have been specified before the $CA command).

A line may be set with zero leading by punching the command $CN(RET). This command will cause the text to be protographed as specified, but no film advance will occur at the end of the line.

The $CN command can be punched anywhere within the line of text to which it applies. It will cancel the stored leading for that line only. But, only the stored leading is cancelled for that line; any $CA command occurring in the same line will cause a film advance. After that line is photographed, the previous stored leading goes back into effect.

2.2.5 Selecting Type Face, Point Size, Column Width, Leading, Set Size, Simultaneously

All five parameters listed above may be selected simultaneously by punching the one command $CCv,w,x,y,z (RET). The five numbers, v,w,x,y,z must be in the following order:

v	=	type face (a number from 1 to 16)
w	=	point size
x	=	column width (in picas and points)
y	=	leading (a number from 0 to 63)
z	=	set size

The five numbers must be separated by commas.

Example:

$CC4,12,11.0,24,12 would specify type face 4, 12 point type, column width 11 picas, 24 points of leading, and set size 12.

One or more of the five numbers may be omitted, but the comma following the omitted number must be inserted (except for numbers omitted at the end of the command).

Examples:

> $CC2,,11,10(RET) This command will cause the type face to be changed to type face 2. Since the number for point size was omitted, the point size will not change. The column width will be changed to 11 picas, the stored leading will be changed to 10 points. Since the number for set size was omitted, a set size will be calculated, corresponding to the new type face and unchanged point size.
>
> $CC,,10.8,13,8(RET) Since the type face and point size were omitted, neither will be changed. The column width will be changed to 10 picas 8 points, the stored leading will be changed to 13 points, and the set size will be changed to 8.
>
> $CC,12,,13,14(RET) The type face was omitted and will not be changed. The point size will be changed to 12. The column width was omitted and will not be changed. The stored leading will be changed to 13, the set size will be changed to 14.

If type face or point size is specified and the set size is omitted, the system will calculate the appropriate set size corresponding to the type face and point size.

If the point size is specified and the leading is omitted, the stored leading will be set equal to the point size.

Example:

> $CC2,8,11(RET) The type face will be set to type face 2, the point size will be set to 8, and the column width will be set to 11 picas. Since the point size was specified, but the leading and set size were not, the leading will be changed to 8 points and the appropriate set size will be calculated.

2.3 INDENT COMMANDS

Indenting may be achieved by use of the commands in the I-group for column indents and hanging indents, the L-group for length fit indents, and the S-group for skew (triangular) indenting. It is important to note that all four types of indents are additive. For example, if a length fit specifies a left indent of 2 picas, this means 2 picas in addition to any left column indent, left hanging indent, etc., which is in effect at the same time.

2.3.1 Column Indents

Left and right indents may be specified by punching the command $ICx,y(RET) where x is the amount of indent (in picas and points) from the left side and y is the amount of indent (in picas and points) from the right side. The two amounts are separated by a comma. The picas and points are separated by a period. If the number of points is zero, both the period and the number of points may be omitted. If the indent is given in points only, the number of picas may be omitted, but the period must be used.

Examples:

> $IC3,2.1(RET) will produce copy indented 3 picas on the left and 2 picas 1 point on the right.
>
> $IC1.6,.11(RET) will produce copy indented 1 pica 6 points on the left and 11 points on the right.

If either the left or right indent is to be zero, it may be omitted from the command. However, if the left indent is omitted, the comma must still be inserted.

Examples:

> $IC3(RET) will produce copy indented 3 picas on the left and no column indent on the right.
> $IC,2.6(RET) will produce copy indented 2 picas 6 points on the right, and no column indent on the left.

All text following a $IC command will be set with the specified indents until a new $IC command is punched or until a $IR command is punched (see below). If a $IC command is punched in the middle of a line of text, it will not become effective until the following line; the current line will be set with the old indent.

The indents specified by $IC commands are added to any hanging indents, length fit indents, or skews which are in effect at the same time.

Column indents can be removed by punching the command $IR(RET). This command sets both the left and right indents to zero. If a $IR command is punched in the middle of a line of text, it will not be effective until the following line; the current line will be set with the old indents.

2.3.2 Hanging Indents

Hanging indents may be specified by punching the command $IHw,x,y,z(RET) where

w	=	left indent in picas and points
x	=	right indent in picas and points
y	=	number of lines to wait before beginning the indent
z	=	number of lines to indent

If y = 0, the indents are effective immediately. If z = 0, the indents are to continue indefinitely, or until stopped by a $IS command (see below).

The four numbers must be separated by commas. If any of the four numbers is omitted, it is assumed to be zero (the commas cannot be omitted, except at the end of the command).

Examples:

> $IH1.6,1,2,4(RET) would produce copy in the following form:

⬅⬅⬅⬅⬅⬅⬅⬅⬅⬅⬅ **20 Picas** ➡➡➡➡➡➡➡➡➡➡➡

Frequently, the user will find that there is
more than one way to specify a desired
output. Often, the choice is one of
personal taste. By efficient use of the
controls, however, the amount of
time, re-punching, and paste-up in-
volved in setting copy can be kept to a
minimum. The key to the operation of the
system lies in properly specifying the con-
trols on the input tape.

Two lines would be set without the indents, then the next 4 lines would be set with a left indent of 1 pica 6 points and a right indent of 1 pica. The rest would be set without the indents.

$IH2.6,,1Ⓡ would produce copy in the following form:

⬅⬅⬅⬅⬅⬅⬅⬅⬅⬅⬅ **20 Picas** ➡➡➡➡➡➡➡➡➡➡➡

Frequently, the user will find that there is
more than one way to specify a desir-
ed output. Often, the choice is one of
personal taste. By efficient use of the
controls, however, the amount of
time, re-punching, and paste-up in-
volved in setting copy can be kept to
a minimum. The key to the oper-
ation of the system lies in properly
specifying the controls on the input
tape.

One line would be set without the indents. The next lines would be set with zero indent on the right and a left indent of 2.6 picas. Since the fourth number in the command was omitted, it is assumed to be zero, so the indents continue indefinitely.

Several hanging indents can be strung together in the same command, separated by semi-colons, with a (RET) punched only at the end of the command. A maximum of six hanging indents can be strung together in one command. Each portion of the command is satisfied in turn, before the next one goes into effect.

Example:

$IH2,2.6,,3;3,,2,4(RET) would produce copy in the following form:

⬅⬅⬅⬅⬅⬅⬅⬅⬅⬅⬅ 20 Picas ➡➡➡➡➡➡➡➡➡➡➡

Frequently, the user will find
that there is more than one way
to specify a desired output. Of-
ten, the choice is one of personal taste. By
efficient use of the controls, however, the
amount of time, re-punching, and
paste-up involved in setting copy
can be kept to a minimum. The key
to the operation of the system lies
in properly specifying the controls on the
input tape.

The first portion of the command says to wait zero lines (in other words, start indenting immediately), then indent the next 3 lines 2 picas on the left and 2 picas 6 points on the right, then return to previous column width. After this has been done, the second part of the command says to wait 2 lines, then indent the next 4 lines 3 picas on the left and no indent on the right, then return to previous column width.

The indents specified by $IH commands are added to any column indents, length fit indents, or skews which are in effect at the same time.

Punching a new $IH command will wipe out any previous $IH command and replace it with the new one. If a $IH command is punched in the middle of a line of text, it will not go into effect until the following line.

A hanging indent may be stopped by punching the command $IS(RET). This command will remove the entire $IH command, regardless of how much of the command has been processed. If a $IS command is punched in the middle of a line of text, hanging indents will not be removed until the following line.

2.3.3 Length Fits

It is possible to specify that a left or right indent be in effect for a given vertical length in picas and points (rather than a given number of lines) to produce a rectangular cut. This may be done by punching a $LL command for left indents, or a $LR command for right indents. We shall discuss the $LL command, since the $LR command is exactly the same for right indents.

The format of the command is $LLx,y(RET) where x is the amount of left indent (in picas and points), y is the length (in picas and points) for which the indent is to stay in effect, and the two amounts are separated by a comma; i.e. $LL horizontal, vertical(RET). If either number is omitted, it is assumed to be zero. The comma may not be omitted.

Examples:

$LL3.6,6.0(RET) would set 6 picas of text with an indent of 3 picas 6 points on the left, then return to previous column width.

$LL4,6(RET)$LR3,8(RET) would set 6 picas of text indented 4 picas on the left and 3 picas on the right. The next 2 picas would be set with zero indent on the left and indented 3 picas on the right. Then the copy would return to previous column width.

Several length fits can be strung together in the same command, separated by semi-colons, with a (RET) punched only at the end of the command. Each length fit in the command is satisfied in turn, before the next one goes into effect. A maximum of six length fits can be strung together in one command.

Example:

$LL3,4;,6;2.5,3(RET) would set 4 picas with a 3 pica left indent, then 6 picas with no left indent, then 3 picas with a 2 picas 5 point left indent, then return to previous column width.

The indents specified by $LL and $LR commands are added to any column indents, hanging indents, or skews which are in effect at the same time.

Length fit indents may be removed by punching the command $LS(RET). This command will remove both $LL commands and $LR commands, regardless of how much of the commands have been processed. If a $LS command is punched in the middle of a line of text, length fit indents will not be removed until the following line.

2.3.4 Skews (Triangular Indents)

It is possible to produce copy that is skewed either on the left side or the right side by use of a $SL command (for left-sided skewing) or a $SR command (for right-sided skewing). We shall discuss the $SL command, since the $SR command is exactly the same.

The format of the command is $SLx,y(RET) where x is the length (in picas and points) of the horizontal part of the triangle, y is the length (in picas and points) of the vertical part of the triangle. The two amounts are separated by a comma. A normal skew would be printed in the following form:

Frequently, the user will find that there is more
than one way to specify a desired output.
Often, the choice is one of personal
taste. By efficient use of the controls,
however, amount of time,
re-punching, and paste-up in-
volved in setting copy can be kept to a min-
imum.

It is also possible to produce a reverse (upside-down) skew by punching a hyphen (-) before the x in the command.

Examples:

$SL-6, 6 (RET) would produce copy in the form:

```
                    Frequently, the user will find
                 that there is more than one way
              to specify a desired output. Often,
           the choice is one of personal taste. By
        efficient use of the controls, however,
     amount of time, re-punching, and paste-up
involved in setting copy can be kept to a
minimum.
```

where the horizontal length of the triangle is 6 picas and the vertical length is 6 picas.

$SL6, 6 (RET) would produce copy in the form:

```
Frequently, the user will find that there is more
   than one way to specify a desired output.
      Often, the choice is one of personal
         taste. By efficient use of the controls,
            however, amount of time,
               re-punching, and paste-up in-
volved in setting copy can be kept to a min-
imum.
```

where the horizontal length of the triangle is 6 picas and the vertical length is 6 picas.

$SL6, 6 (RET) $SR6, 6 (RET) would produce copy in the form:

```
Frequently, the user will find that there is more
   than one way to specify a desired output.
      Often, the choice is one of person-
         al taste. By efficient use of
            the controls, how-
               ever, amount
of time, re-punching, and paste-up involved in
setting copy can be kept to a minimum.
```

where the horizontal length of each skew is 6 picas and the vertical length of each skew is 6 picas.

Several skews can be strung together in the same command, separated by semi-colons, with a (RET) punched only at the end of the command. Each skew in the command is satisfied in turn, before the next one goes into effect. A maximum of six skews can be strung together in one command.

The indents produced by a skew command are added to any column indents, hanging indents, or length fit indents which may be in effect at the same time.

Skews may be stopped by use of the command $SS(RET). This command will remove both $SL commands and $SR commands, regardless of how much of the commands has been processed. If a $SS command is punched in the middle of a line of text, the skew indents will not be removed until the following line.

2.4 TABULAR COMMANDS

Tab stops can be set up by using the commands in the T-group. In most cases, tab stops should be set up before punching the line of text for which the tab stops are needed.

A change in column width or an indent specification will not change the positions of the tab stops. Care should be taken if the new column width is so narrow that all of the tab stops do not fall within the column; the results of setting tabular text in this case are unpredictable.

It should be remembered that T-group commands merely set up tab stops. They do not cause text to be set in a tabular format. Setting text in relation to tab stops is done by punching tab keys (see Section 2.9.2).

2.4.1 Setting Specific Tab Stops

Tab stops may be set at any specific location within the column by punching the command $TSx,y,...,z(RET) where x,y,...,z are the distances (in picas and points) from the left margin.

Example:

> $TS2,5.6(RET) would set two tab stops, one at 2 picas from the left margin, the other at 5 picas 6 points from the left margin.

A maximum of 18 tab stops may be set with this command. The tab stops set up by this command will stay in effect until another T-group command is punched, at which time the tab stops will be wiped out and the new tab stops will be set up.

2.4.2 Setting Equally-Spaced Tab Stops

Tab stops may be set up equally-spaced across the column by punching the command $TCx(RET) where x is a number from 1 to 19 indicating the number of sub-columns to be set up within the column.

Examples:

> $TC4(RET) would set three tap stops, producing 4 equal sub-columns across the column. For example, if the column width had been defined as 14 picas, tab stops would be set at 3.6, 7.0 and 10.6.
>
> $TC2(RET) would set one tab stop in the middle of the column.
>
> $TC1(RET) would set no tab stops.

A maximum of 19 sub-columns may be set up by a $TC command. The tab stops set up by this command will stay in effect until another T-group command is punched, at which time the tab stops will be wiped out and the new tab stops will be set up.

Remember that a change in column width will not change the positions of the tab stops. Therefore, if the column width is changed, the tab stops set up by the previous $TC command will no longer define equal sub-columns.

2.4.3 Setting Proportionately-Spaced Tab Stops

Tab stops may be set up proportionately-spaced across the column by punching the command

$TPx,y,...,z(RET) where x,y,...,z are numbers indicating the ratios of the sub-columns to each other.

Examples:

> $TP1,1(RET) would set up two sub-columns (one tap stop) in the ratio 1:1 (equal sub-columns).
>
> $TP2,1(RET) would set up two sub-columns in the ratio 2:1 (the first sub-column twice as wide as the second one). If the column width had been defined as 12 picas, the tab stop would be set at 8 picas.
>
> $TP2,2,3,5(RET) would set three tab stops, producing four sub-columns in the ratio 2:2:3:5. If the column width had been defined as 12 picas, the tab stops would be set at 2.0, 4.0, 7.0.

The tab stops set up by this command will stay in effect until another T-group command is punched, at which time the tab stops will be wiped out and the new tab stops will be set up.

A change in column width will not change the positions of the tab stops. Therefore, if the column width is changed, the tab stops set up by the previous $TP command will no longer define the proportionate sub-columns which were intended.

2.4.4 Setting Tab Stops Determined by Text

A tab stop may be set at a point on the line determined by the position of the text. This can be done by punching the command $TT(RET). This command sets a tab stop at that point on the line where the command is encountered.

Examples:

> ABC$TT(RET)(QL) would set ABC flush left and set a tab stop immediately following the C.
>
> ABC$TT(RET)DEF(QL) would set ABCDEF flush left and set a tab stop immediately following the C (at the position where D is).
>
> $TT(RET)ABC(QR) will set ABC flush right and set a tab stop at the position where A is.

A $TT command can only be used in lines which are set by quad or tab keys. The command cannot be used in a line which is being automatically justified by the PDP-8, because the possible expansion of inter-word spaces is not taken into account.

The $TT command is the only T-group command which does not wipe out other tab stops already existing. A tab stop set up by a $TT command is added to the tab stops which already exist. The tab stops set up by $TT commands stay in effect until removed by another T-group command.

2.4.5 Removing Tab Stops

Tab stops may be removed at any time by punching the command $TR(RET). This command will immediately remove all tab stops set up across the column.

2.5 OUTPUT FORMAT COMMANDS

Certain other features of the printed result may be specified by commands in the O-group. These features include specifying ragged output, hyphenation, letter-spacing, and others.

NOTE

> When punching these commands, it is important that the $ be followed by the letter O, not the number zero.

2.5.1 Ragged Formats

a. Ragged left format can be specified by punching the command $OL(RET). This causes all following lines of text to be set flush right, with minimum inter-word spacing and no letter-spacing. Lines are not justified. The output will continue to be ragged left until the next time that a quad or tab key is punched, at which time output will return to normal.

Frequently, the user will find that
there is more than one way to specify
a desired output. Often, the choice is
one of personal taste. By efficient use
of the controls, however, amount of
time, re-punching, and paste-up in-
volved in setting copy can be kept to a
minimum.

b. Ragged right format may be specified by punching the command $OR(RET). This causes all following lines of text to be set flush left, with minimum inter-word spacing and no letter-spacing. Lines are not justified. The output will continue to be ragged right until the next time that a quad or tab key is punched, at which time output will return to normal.

Frequently, the user will find that
there is more than one way to specify
a desired output. Often, the choice is
one of personal taste. By efficient use
of the controls, however, amount of
time, re-punching, and paste-up in-
volved in setting copy can be kept to a
minimum.

c. Ragged center format can be specified by punching the command $OC(RET). This causes all following lines of text to be centered, with minimum inter-word spacing and no letter-spacing. Lines are not justified. The output will continue to be ragged center until the next time that a quad or tab key is punched, at which time output will return to normal.

Frequently, the user will find that
there is more than one way to specify
a desired output. Often, the choice is
one of personal taste. By efficient use
of the controls, however, amount of
time, re-punching, and paste-up in-
volved in setting copy can be kept to a
minimum.

d. Ragged middle format can be specified by punching the command $OM(RET) or $OMx(RET) where x is a number. Ragged middle format is staggered ragged center, producing copy in the following form:

Frequently, the user will find that
there is more than one way to
specify a desired output. Often,
the choice is one of personal taste.
By efficient use of the controls,
however, amount of time,
re-punching, and paste-up in-
volved in setting copy can be kept
to a minimum.

In other words, ragged middle is like ragged center, but the lines are not centered in the middle of the column; the first line is centered a little to one side of the middle, the next line is centered to the other side of the middle, and so on, back and forth.

If the command $OMx(RET) is used, the number x is the amount of stagger (in points); that is, the amount of displacement from the center. If no number x is used in the command, the amount of stagger specified in a previous $OMx(RET) command will be used. If there was no previous specification, the system's standard value will be used.

After a $OM command, the output will continue to be ragged middle, until the next time that a quad or tab key is punched, at which time output will return to normal.

2.5.2 Hyphenation

The command $OH(RET) or $OHx(RET) where x is a number can be used to indicate that the text may be hyphenated in order to produce justified copy. If the number x is used, it is the word spacing (in relative units) at which hyphenation should be attempted (i.e. the maximum amount of word spacing that may be used in trying to justify a line before hyphenation is attempted). If no number x is used in the command, the amount of word spacing specified in a previous $OHx(RET) command will be used. If there was no previous specification, the system's standard value will be used.

The command $ON(RET) is used to indicate that the text may not be hyphenated. Hyphenation will not be attempted again until a $OH command is punched.

The PDP-8 Photon 513-1560 System assumes that text is hyphenable unless otherwise designated. Therefore, if the output text can be hyphenated, a $OH command is not necessary, except to specify a word-spacing amount, or if a $ON command was previously used.

There is more to be considered about hyphenation. The manner in which the PDP-8 hyphenates words is a two-step process. Step 1 is to check through a special list of words to see if the word to be hyphenated is in the list. If it is, the computer immediately knows how to hyphenate the word. If the word is not in the list, the computer goes on to the second step. Step 2 is to use a set of logical rules to hyphenate the word. This second step can be a lengthy process.

To save time, it may sometimes be desirable to omit this second step of the hyphenation process. This can be done by punching the command $OI(RET). This command allows the computer to try to hyphenate (when necessary) using Step 1 only. If a word cannot be hyphenated by Step 1, no further attempt is made to hyphenate the word.

The $OI command will stay in effect until it is cancelled by punching the command $OU(RET). The $OU command re-enables the use of the logical hyphenation rules. The PDP-8 Photon 513-1560 System always assumes that the logical hyphenation rules may be used, unless otherwise specified. Therefore, it is never necessary to punch a $OU command unless a $OI command has been previously punched.

2.5.3 Letter Spacing

The command $OS(RET) or $OSx(RET) or $OSx,y(RET) (where x and y are numbers) can be used to indicate that letter-spacing may be used in justifying the lines of text. If the number x is used, it is the word spacing (in relative units) at which letter-spacing should be attempted (i.e., the maximum amount of word spacing that may be used in trying to justify a line before letter spacing is used). If the number y is used, it is the maximum amount of inter-letter space (in relative units) which can be used in trying to justify a line. If either x or y (or both) is not used in the command, the amounts of word spacing and letter spacing specified in a previous $OS command will be used. If there was no previous specification, the system's standard values will be used.

The command $OW(RET) is used to indicate that the text cannot be letter-spaced. Letter spacing will not be used again until a $OS command is punched.

The PDP-8 Photon 513-1560 System assumes that text may be letter-spaced unless otherwise designated. Therefore, if the output text can be letter-spaced, a $OS command is not necessary, except to specify a word-spacing or letter-spacing amount, or if a $OW command was previously used.

2.5.4 Specifying Minimum Word Space

The minimum word space that may be used in justifying a line can be specified by punching the command $OJx(RET) where x is the minimum word space (in relative units). If no $OJ command is punched, the system's standard value will be used.

2.5.5 Specifying No Flash

The command $OF(RET) is used to specify that no flash be given for the current line. This will cause the Photon 513-1560 carriage to move exactly as though it were setting the text, but none of the text will be photographed. This command is useful when it is necessary to space out the carriage to a point determined by the position of the text but the text itself is not to be set until a later line. (It may be necessary to space out the carriage in order to set a tab stop at that point using the $TT(RET) command.) The $OF command remains in effect until a quad or tab key is punched.

Example:

```
$CN(RET)ABC$TT(RET)$OF(RET)(QL)(TL)abc(TL)ABC(QL) would produce:

                 abc
             ABC
```

In other words, the carriage spaces out to the point where ABC(QL) would end, and a tab stop is set there. The (QL) key causes the $OF command to end, so the following text is photographed. The rest of the above example causes abc to be set at this new tab stop, and then ABC is set on the following line.

If is important to note that when a quad key is encountered after a $OF command, the quad key will be treated as a normal end-of-line, and the film will be advanced according to the stored leading. If it is desired to have no film advance, a $CN command must be given for the line, or else the stored leading must be changed to zero.

2.6 HOLD COMMANDS

H-group commands differ somewhat in format from the commands belonging to the other command groups. In punching the commands, the $H is followed by a number (from 0 to 9), then followed by another letter. These commands are used to store text temporarily and then to bring the text back into the processing later on.

2.6.1 Saving Text

Up to 125 characters of text may be read from the input tape and saved for later use by punching the command $HxSabc...wyz(RET) where x is a number from 0 to 9 and abc...wyz is the text to be stored. (Note that (RET) is punched after the text to be stored.)

Example:

$H2SHello!(RET) will have the following effect:

The text "Hello!" will be stored away for future use, and the PDP-8 will continue reading and processing the input tape as though that text had never appeared; that is, no space will be reserved for it on the current line being assembled.

Only text may be saved by this command; other commands or controls cannot be saved. If any commands or special keys are mixed in with the text to be stored, an error will occur and the entire $HxS command will be ignored.

It is possible to save ten different pieces of text by using $HxS commands with different numbers for x (from 0 to 9). For example, after having used $H2SHello!(RET) to save "Hello!", we could also use $H3SGoodbye(RET) to save "Goodbye", without wiping out "Hello!".

Once text has been stored away by a $HxS command, it remains in the system permanently, until another $HxS command using the same number x is punched. When this happens, the text previously stored is removed and replaced by the new text. In the example above, punching $H2SGoodbye(RET) would cause "Hello!" to be wiped out and replaced by "Goodbye".

2.6.2 Recalling Text

Any text that was saved by a $HxS command can be recalled at any time by punching the command $HxR(RET) where x is the same number that was assigned to that text in the $HxS command when the text was stored. Recalling the saved text causes the computer to treat the text just as though it had been punched on the input tape at the place where the $HxR command is punched.

Example:

> Suppose the command $H2SHello! (RET) had previously been punched. Punching $H2R(RET)(QL) would produce:
>
> Hello!

Recalling saved text does not cause the text to be wiped out of the system. The same text may be recalled over and over.

2.7 VISUAL COMMANDS

It is possible to have the computer punch out messages on the output tape which are readable to the human eye. This can be done by using commands in the V-group. This is particularly useful for punching out information at the beginning of an output tape which will serve to identify the output tape.

2.7.1 Punching Visuals

A message may be punched on the output tape by use of the command $VPabc...xyz(RET) where abc...xyz is the message to be punched. The message may consist of any letters, numbers, the hyphen and the space bar, but no other characters. The message must not contain more than 125 characters in all.

Example:

> $VPMONDAY NOV 7(RET) would cause "MONDAY NOV 7" to be punched on the output tape as soon as the command is encountered on the input tape.

Note that in punching the command, the (RET) is punched after the message. The message will not appear as part of the printed output.

2.7.2 Punching Visual and Tag Lines

A message may be both punched on the output tape and photographed by use of the command $VTabc...xyz (RET) where abc...xyz is the message. The message may consist of any letters, numbers, the hyphen and the space bar, but must not contain more than 125 characters in all.

The message is photographed flush left, and then 72 points of leading is added. This creates a tag line for identification purposes. However, the message will not be photographed unless type face, point size, column width, and leading have been specified.

Since the lines and the leading produced by this command are counted by the system, when this command is used it must be punched before any hanging indent, length fit or skew commands are punched.

2.8 DEFINING NEW COMMANDS

Since some commands are rather lengthy, it is convenient to have a shorter way of specifying those commands which will be used frequently. This is done by defining new commands to perform the same functions as the longer basic commands. For example, suppose the command $LR6,6;,6;6,4(RET) is to be used frequently. This is cumbersome to punch repeatedly, and it would be convenient to punch a shorter command, say $LX(RET), and have it mean the same thing. This can be done if $LX(RET) is first defined to mean the same thing as the longer command.

2.8.1 Defining Commands

Each command group, except the H-group, has a defining command. It is as follows: First a $ is punched. Next punch the letter specifying the command group to which the longer command belongs. In the example above, the letter is L. Next punch a D (meaning Define). Next, another letter which is not used as a second letter in any other command belonging to that command group is punched (in the example above, X is not used as the second letter in any L-group command, so we may use X for the new command). Finally, punch the rest of the longer command beginning with the second letter, all the way through to the (RET).

What has been punched is: LDX6,6;,6;6,4(RET) where L is the group to which the command belongs, D means defining command, X is the new letter assigned to the command, R6,6;,6;6,4 is the command we are replacing and (RET) means end of defining command. From now on whenever the command $LR6,6;,6;6,4(RET) is needed, the command $LX(RET) can be punched instead (X is the letter assigned to the new command in the defining command), and the longer command will be executed. Thus, we now have a new command—a $LX command.

Further examples:

> Suppose the command $TS3.0,6.6,9.6(RET) is needed frequently to set tab stops. Define a new command by picking any letter not used as the second letter in any other T-group commands, say B. Punch the defining command as follows: $TDBS3.0,6.6,9.6(RET) where

T	=	T-group
D	=	defining command
B	=	letter assigned to new command
S	=	letter of original command

From now on, punching the command $TB(RET) will cause tab stops to be set at 3 picas, 6 picas 6 points, and 9 picas 6 points.

Suppose also the command $TS4.0,7.0(RET) is needed frequently. We can define a new command for this using a different second letter, say F. Then the command $TDFS4.0,7.0(RET) will define the new command $TF, and then punching $TF(RET) will cause tab stops to be set at 4 picas and 7 picas.

It is important that the letter assigned to the new command be one that is not used in any of the basic commands described earlier in this handbook. Otherwise, an error will occur and the new command will not be defined. See Appendix II for a list of all basic commands in each command group. In each group, any letter not used in one of the basic commands can be used to define a new command.

A defining command need be punched only once. The new command defined by it then becomes a permanent part of the system and stays in effect for each new input tape. The defined command can be removed only by defining a new command for that letter or by punching a deleting command. The defining commands can be included as part of an input tape, or they can be punched on a separate tape and fed into the computer.

2.8.2 Deleting Commands

Each command group, except the H-group, has a deleting command to remove defined commands. The deleting command is as follows: First a $ is punched, then the letter specifying the command group to which the command belongs is punched. Next a Z (meaning delete) is punched. Next the letter assigned to the command that is to be removed is punched. Finally a (RET) is punched. In one of the examples above, we defined a command $LX(RET) to equal the command $LR6,6;,6;6,4(RET). Suppose this command will no longer be needed. The $LX command can be removed by punching $LZX(RET) where

L	=	L-group
Z	=	deleting command
X	=	command to be deleted

From now on, punching $LX(RET) will result in an error unless a new definition is given to $LX, because the $LX command has been removed from the system.

None of the basic commands listed in Appendix II can be removed by a deleting command. If an attempt is made to delete one of the basic commands an error will occur.

Deleting commands can be included as part of an input tape, or they may be punched on a separate tape and fed into the computer.

2.9 DUMMY COMMANDS

Each command group except the H-group contains a dummy command of the form $xY(RET) where x is the letter specifying the command group to which the command belongs. For example, the I-group dummy command is $IY(RET), the T-group dummy command is $TY(RET). The meaning of this command is "do nothing", i.e., ignore the command. The use of this command can best be understood by looking at an example:

> Suppose some text is to be set, and we are not certain whether it will look better set ragged center or set with automatic justification. We could first set it ragged center by punching the command $OC(RET) on the input tape before the text is punched, then punching all the text, along with the necessary commands and controls. If, after seeing the printed output, we decide that it would look better if the text had been justified, the only way to reset the text would be to punch the entire tape again, leaving out the $OC command.
>
> But suppose instead that we define a new command, say $OA(RET), to be equivalent to $OC(RET) (the defining command $ODAC(RET) can be punched on a separate tape and read into the PDP-8). Then we punch the input tape in the normal way, except instead of punching $OC(RET), we punch $OA(RET). The text will be set ragged center. Now, if we decide we don't want it set ragged center, we do not have to re-punch the input tape. We simply give a new definition to $OA(RET); namely, we define it to be the dummy command. (The new defining command $ODAY(RET) can be punched on a separate tape and read into the PDP-8.) This time, $OA(RET) will not mean ragged center. It will mean "nothing"; in other words, the command $OA(RET) will be ignored, and the text will be set with normal justification.

2.10 SINGLE-KEY FUNCTIONS

As was mentioned earlier, some functions can be accomplished by punching a single key at the appropriate place in the text. These include such operations as punching a (QL) key at the end of a line which is to be set flush left, or punching a (WL) key at the point in the text where leaders are to be inserted. These single-key functions are described below.

2.10.1 Use of Quad Keys

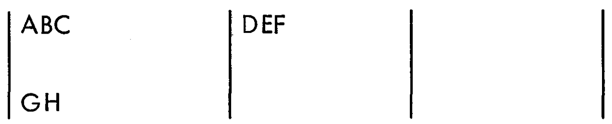
The keys, quad left (QL), quad right (QR), and quad center (QC) are all similar. If no tab stops have been set up, punching one of these keys will cause all preceding text being assembled on the current line to be set at the left, right, or center of the column. In other words, punching (QL) will cause the current line to be set flush left, with minimum word spacing and no letter spacing. Punching (QR) will cause the current line to be set flush right, with minimum word spacing and no letter spacing. Punching (QC) will cause the current line to be centered, with minimum word spacing and no letter spacing.

All text following one of these quad keys will be set on the next line — punching one of these quad keys immediately ends the line.

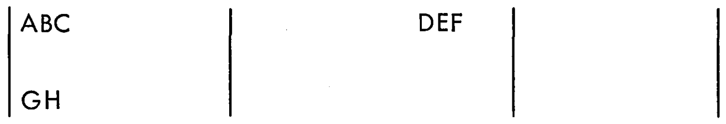
If tab stops have been set up, these quad keys will perform their same functions, but only on the remainder of the column (rather than the full column). For example, if some text had already been set on the line by a tab key, punching more text followed by a (QL) would set the second bit of text flush left at the tab stop, not at the left margin. If the second bit of text had been followed by a (QC), the text would have been centered in the remainder of the column (from the tab stop to the right margin), not the full column. Punching (QR) always sets the text at the right margin.

Examples:

Assume that two tab stops have been set up. Punching ABC(TL)DEF(QL)GH(QL) would produce:



Punching ABC(TL)DEF(QC)GH(QL) would produce:



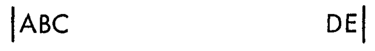
where DEF is centered between the first tab stop and the end of the column.

Even if tab stops have been set up, punching one of these quad keys will immediately end the line.

It is possible to set text flush left and right, with all the blank space in the middle. This is done by punching a (QM) at the point in the text where the break is to occur, and a (QR) or (TR) at the end of the text which is to be set flush right. The text preceding the (QM) is set flush left, with minimum word spacing and no letter spacing, but the line is not ended. The carriage is advanced to the point in the text where the (QM) appears, and processing continues with the text following the (QM).

Example:

Punching ABC(QM)DE(QR) would produce:



Again, if tab stops have been set up, punching (QM) will cause the preceding text to be set flush left within the current sub-column.

(QM) is the only quad key which does not cause the current line to end.

2.10.2 Use of Tab Keys

After tab stops have been set up by a T-group command, the sub-columns may be treated as separate columns for the purpose of justification or the setting of text left, right, or center within the sub-columns. With the exception of the (TA) key, each time a tab key is punched, the specified operation is performed and the next bit of processing is assumed to apply to the next sub-column (rather than to the next line). Within a sub-column we may tab left, right, or center by punching (TL), (TR), or (TC) following the text to be set in that sub-column. Any text following one of these tab keys will be set in the next sub-column, unless otherwise specified.

Examples:

Assume the column has been divided into three sub-columns. Punching AB(TL)CDE(TL)FG(TL) would produce:

```
|AB          |CDE          |FG           |
```

Punching AB(TL)CDE(TR)FGH(TC) would produce:

```
|AB             |        CDE|              FGH
```

Punching AB(TL)(TL)CDE(TL) would produce:

```
|AB                                        |CDE
```

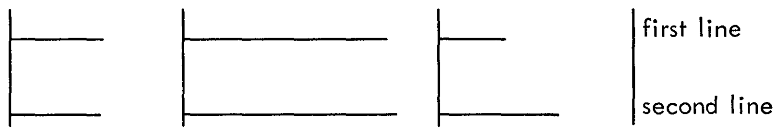
Punching AB(TR)CDE(TL)FG(TR)HIJ(TL) would produce:

```
              AB|CDE        |            FG|
|HIJ            |           |              |
```

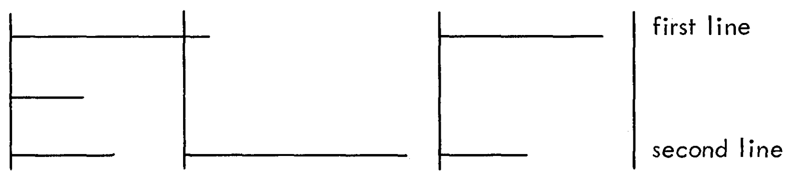
since after the last sub-column on the line, output returns to the first sub-column of the next line.

In the last example, three sub-columns had been defined, and after three tab keys had been punched, the line was ended and processing continued on the next line. This will always happen when a tab key has set text in the last sub-column. However, as a precaution, it is much better to end the line with a quad key. That way, if a tab key has been forgotten in the middle of the line, only one line will be wrong. The next line will begin with the text with which it is supposed to begin. In other words, the last example would have been better if it had been punched as AB(TR)CDE(TL)FG(QR)HIJ(TL). The output produced by this would have been the same.

The situation may arise in which text to be set within a sub-column is wider than the sub-column. This will result in one line being set improperly, after which output will return to the desired form. For example, suppose two lines of text are to be set as follows:



Assume, however, that the text intended for the first sub-column of the first line is too wide for the sub-column. The printed result will be as follows:

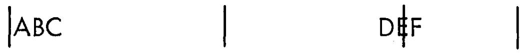


The first line is printed improperly, requiring two lines to furnish three sub-columns of text, due to the overlap of the first sub-column. However, the second line (and all lines following) will be printed as intended.

Punching a (TA) causes preceding text to be centered on the next tab stop (not counting the left and right margins as tab stops). That is, text is centered on the tab stop which is to the right of the sub-column.

Example:

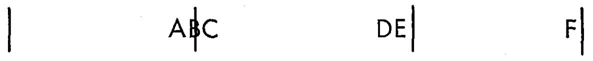
ABC(TL)DEF(TA)(QL) would produce:



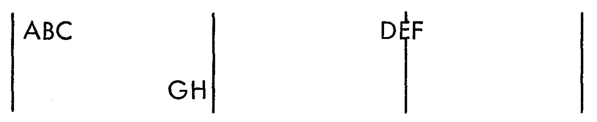
There is an important difference between (TA) and the other tab keys. After the text preceding the (TA) key is set, the carriage advances only to the last character of the text, not to the next sub-column.

Examples:

If two tab stops were set up, punching ABC(TA)DE(TR)F(QR) would produce:



Punching ABC(TL)DEF(TA)(QL)GH(TR)(QL) would produce:



If the first (QL) key had been omitted, GH would have been printed on the first line, instead of the second line.

All text set by tab keys will be set with minimum word spacing and no letter spacing, unless the text overflows the right margin. In that case, the System will break the text and justify that part of the text remaining on the line, while printing the rest of the text in the first sub-column of the next line.

2.10.3 Inserting Leaders

Leader dots may be inserted by punching a (WL), the "with leaders" key, followed by a quad or tab key. This will result in the text being set with minimum word spacing according to the specification of the quad or tab key, and the remaining space on the line or sub-column being filled with leader dots.

Example:

ABC(WL)(QC) would produce:

```
|. . . ABC . . .|
```

Any character other than dots can be used to fill the remaining space in a column or sub-column by punching (WX), followed by the character, followed by a quad or tab key.

Example:

ABC(WX)*(TL)45(TL) would produce:

```
|ABC******|45     |
```

The leader dots (or other character) will be set in the same type face and point size as the text which precedes the (WL) or (WX) key. If the leader character is to be set in a different style or size, the necessary C-group command must be punched before the (WL) or (WX).

Examples:

$CC1,12(RET)ABC(QM)(WL)DEF(QR) would produce:

ABC. . . . . . DEF all in the same style and size.

$CC1,12(RET)ABC(QM)$CC2,14(RET)(WL)DEF(QR) would produce the same output, except that while ABC would be 12 point in type face 1, the leader dots and the DEF would be 14 point in type face 2.

$CC1,12(RET)ABC(QM)$CC2,14(RET)(WL)$CC3,12(RET)DEF(QR) would produce the same output, except that ABC would be 12 point in type face 1, the leader dots would be 14 point in type face 2, and DEF would be 12 point in type face 3.

When leadering with a character other than dots, nothing can be punched between the (WX) and the character except a shift or unshift, if necessary.

2.10.4 Use of Justify Key

Punching the justify key, (J), causes the preceding text being assembled on the current line to be set as a justified line.

In other words, if no tab stops have been set up, all text between the end of the previous line and the (J) will be spread out across the column; the extra space will be used up as word spaces (and letter spaces, if necessary). Any text following the (J) will be set on the next line. The (J) is only effective for the line on which it is punched; it has no effect on the following lines.

If tab stops have been set up, punching a (J) will cause all text between the last tab key (or end of the previous line) and the (J) to be spread out across the current sub-column. Any text following the (J) will be set in the next sub-column (rather than the next line). The (J) is only effective for the sub-column in which it occurs; it has no effect on the following sub-columns.

2.10.5 Use of Delete Key

If the perforator operator makes a mistake and realizes it right away, the mistake can be corrected by punching a (DEL). If the mistake occurs in the midst of text, punching a (DEL) will remove everything back to either the last word space, the last quad key, the last tab key, the last (J), or the last hyphen (whichever of these things is found first).

Examples:

SALE(QL)One day om1(DEL)only(QL) would be printed as:

SALE

One day only

In other words, everything back to the last word space was removed (the word space was found before the quad key).

STORE-WED(DEL)WIDE SALE(QL) would be printed as:

STORE-WIDE SALE

Everything was removed back to the hyphen.

If a (DEL) is punched in the middle of a command, the whole command is removed.

2.10.6 Use of Return Key

All of the commands described earlier in this chapter must be ended by punching a (RET). If (RET) is punched at any other time, it will be ignored.

(RET) is never used to mean end of line.

2.10.7 Use of Space Bar

The space bar is used to indicate a word space. Striking the space bar two or more times in succession will result in only one word space being inserted, not two or more.

CHAPTER 3
SYSTEM FORMATS

3.1 DEFINING AND USING FORMATS

A great deal of the time spent in setting advertising copy is wasted in setting ads which have the same format or appearance, but different text. An example of this would be a supermarket's daily ads, which always have the same format, but which advertise different products each day. The PDP-8 Photon 513-1560 System includes a feature which greatly reduces the amount of work involved in setting these ads. This feature is the ability to define and use formats.

The preceding chapter described how to define a new command to replace a longer command. It is also possible to replace a whole string, or several strings, of commands and single-key functions by a shorter command called a format. (By a string of commands, we mean a series of commands punched one after the other.) For example, suppose the string of commands $CC2,12,11(RET)$TS3.0,6.5(RET)$ON(RET) were needed to set up a frequently used format. A format could be defined to be equivalent to all of these commands at once.

To define a format, first $F is punched. Next, a number from 0 to 299 is punched. (This will be the number assigned to the format being defined.) Then a D (for define) is punched. Next, the entire string of commands and single-key functions is punched. Finally, (CALL)(CALL) is punched; two (CALL)'s to indicate the end of the definition. In the example above, the format could be defined by punching $F10D$CC2,12,11(RET)$TS3.0, 6.5(RET)$ON(RET)(CALL)(CALL) where

F	=	format
10	=	the number assigned to the format replacing the given string of commands
D	=	define

The rest is the string of commands as they would be punched if they were being used normally (note that no $'s or (RET)'s can be omitted). Finally, (CALL)(CALL) indicates the end of the format definition.

Format definitions can be punched as part of an input tape, or they can be punched on a separate tape and fed into the PDP-8.

After a format has been defined, the way to use it is as follows: Once during each input tape, the format must be made ready. This is done by punching $F, then the number assigned to the format, then C, then a (RET). In the example above, the format would be made ready by punching $F10C(RET). However, using this command does not cause the string of functions to be executed; it only makes the functions ready to be executed. In order for the functions to be executed, a (CALL) must be punched. Each time (CALL) is punched, the entire string of functions will be executed. The string of functions will remain in a ready state until another format is defined or made ready.

Several strings of functions can be specified with one format. In defining such a format, the strings are separated from each other by a (CALL), and (CALL)(CALL) is punched at the end of the last string. To use a format specifying several strings, again it must first be made ready (by punching $FxC(RET) where x is the number of the format). After it has been made ready, each time (CALL) is punched, one string of functions will be executed. the first time (CALL) is punched, the first string of functions will be executed; the next time (CALL) is punched, the second string will be executed; and so on until the last string. Then if (CALL) is punched again, the first string will again be executed.

NOTE

A format may not contain more than 125 characters.

Once a format has been defined, it becomes a permanent part of the system; it need not be re-defined for each new input tape. The format will remain in the system until it is removed either by defining a new format with the same number, or by punching a format-deleting command. To punch a format-deleting command, first $F is punched, then the number of the format to be deleted, then a Z, then (RET). In the example above, format 3 was defined. Punching $F3Z(RET) would remove that format from the system. Once a format has been deleted, trying to make that format "ready" will result in an error.

There is one restriction on deleting and re-defining formats. Formats numbered from 50 to 299 cannot be deleted or re-defined. This is a safety feature of the PDP-8 Photon 513/560 System; it prevents the accidental removal of those formats which should remain in the system permanently. The formats numbered from 0 to 49 can be deleted or re-defined at will.

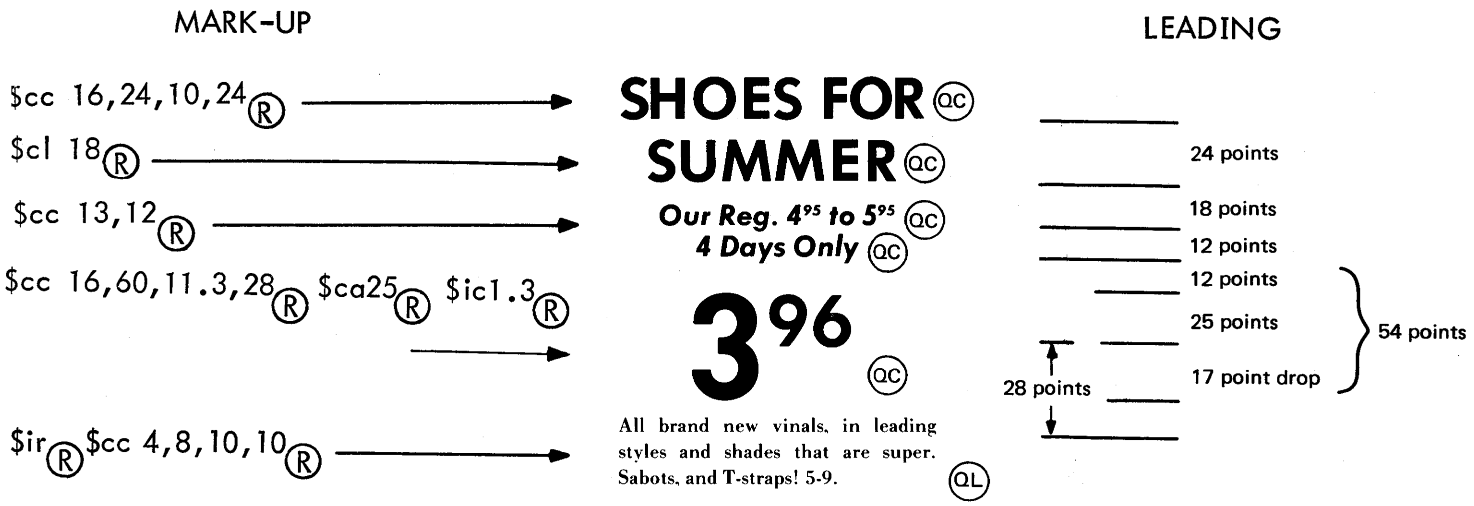
CHAPTER 4

DISPLAY AD MARK-UP AND FORMATTING

MARK-UP

Straight mark-up should be understood before formatting is attempted. Leading often causes problems in measurement of copy because the manner in which it is measured differs (base-line to base-line). Try to mark-up and set the following ad, without the help of the commands marked on the ad before going on to formatting.

The mark-up and formats on the following pages were written for a PHOTON 560 equipped with a single disk containing 16 type faces.



```
MARK-UP

$cc 16,24,10,24Ⓡ
$cl 18Ⓡ
$cc 13,12Ⓡ
$cc 16,60,11.3,28Ⓡ $ca25Ⓡ $ic1.3Ⓡ
$irⓇ$cc 4,8,10,10Ⓡ
```

NOTE: If another ad is to be set following the above ad, be sure to allow enough white space for cutting in by inserting a $ca command here. This will allow 10 points unless more is specified.

HELPFUL HINTS:

1. Stored leading (film advance) is normally set after the current line is flashed unless leading is added with the $ca command. Added leading is immediately set at the point that the $ca command is used in the current line.

2. Large type (60 and 72 point) has a vertical and horizontal deviation from other type sizes. This ad was set with allowances made for a 17 point vertical and 15 point horizontal.

3. The commands circled above represent a single key on a keyboard. This practice helps to avoid confusion for perforators. Interpretation of these commands can be found in Appendix A.

FORMATTING FREQUENTLY USED ADS

Ads that are to be rerun should be formatted and catalogued for future reference by the user. Drawing a block diagram of the ad layout enables the mark-up personnel to remember the ad and communicates to the perforators the order in which the ad is to be typed.

DEPARTMENT STORE AD

One line of 18-point type can be set if the (QC) (quad-center) is omitted when using the format. The $tc2 Ⓡ (tab compute) command divides any column width, specified by the user, into two columns. The (TC) (tab center) and (QC) commands center the 8-point and 48-point text in the columns.

Format Defined:

```
$f2d
$cc3,18,,24 Ⓡ
        Ⓒ
(QC) $cc,8,,10 Ⓡ $ca2 Ⓡ $tc2 Ⓡ
        Ⓒ
(TC)(QL)
        Ⓒ
(TC) $ca8Ⓡ $cc,48,,24Ⓡ
        Ⓒ
(QC) $cc4,12 Ⓡ $tr Ⓡ
        Ⓒ
(QL) $ca36 Ⓡ
        Ⓒ
        Ⓒ
```

Format Used:

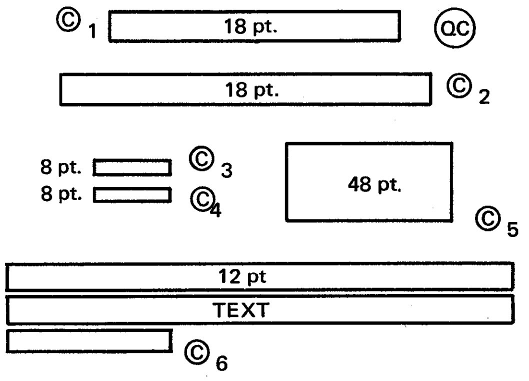
```
$cw16 Ⓡ
$f2c Ⓡ
        Ⓒ
Stainless-Faced
        (QC)
Combination Windows
        Ⓒ
Regular
        Ⓒ
$24.95
        Ⓒ
21.66
        Ⓒ
Requires little or no upkeep
Choose . . . . etc.
        Ⓒ
        Ⓒ
Start of next ad (first line)
        (QC)
```

Stainless-Faced
Combination Windows

Regular
$24.95

21^{66}

Requires little or no upkeep. Choose the finish that suits the needs of your home!



HELPFUL HINTS:

1. It is useful, in most cases, to omit column width when defining a format. Typing $cw16 ® before calling the format changes that format to any column width desired.

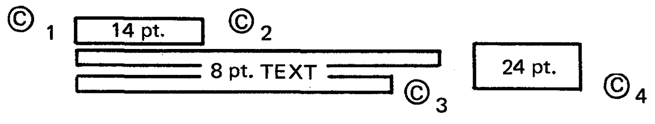
2. When defining a format, two call keys© © punched together signify the end of the format. When using a format, two ©'s punched together usually mean that the format has been restarted. However, page 4-5 illustrates an example of two © 's together indicating that text has been left out.

3. The $ca36 ® command increases the leading to 48 points between the end of the first ad and the beginning of the second ad to prevent oversetting.

AN AUTOMOBILE AD

Automobile ads often consist of a few lines or a block of justified text with the last line leadered out to a price. An extremely versatile format for this type of ad is explained below.

1966 FORD
Convertable with white wall tires, radio, heat-er, all the extras, plus low milage................. **$1588**
1967 CHEVY
New interior with bucket seats, plus v8 engine, *radio and heater, plus new tires*................... **$1775**



Format Defined:

```
$f3d
$cc8,14,,8 ®
          ©
(QL) $cc14,8,,8 ® $ic,3.6 ®
          ©
(QM) (WL) $cn ® (QR) $ir ®
$cc8,24,,14,18 ®
          ©
(QR)
          ©
          ©
```

Format Used:

```
$cw16 ®
$f3c ®
          ©
1966 Ford
          ©
Convertible with white wall
tires, radio, heater, all the
extras, plus low mileage
          ©
$1588
          ©
          ©
1967 Chevy
   (Etc.)
```

HELPFUL HINTS:

1. The 8-point text is right-justified to a column indent of 3.6 picas to make this format adaptable to any change in column measure.

2. The $cn® (cancel lead) command prevents the PHOTON 513 and 560 from leading after the last line of 8-point text is set. The quad right command removes this condition for the succeeding line; the indent is removed, and the 24-point is set flush-right and base aligned with the 8-point.

3. To present a better appearance and allow more room for text, the 24-point price is "kerned" by changing its set size from 20.5 to 18.

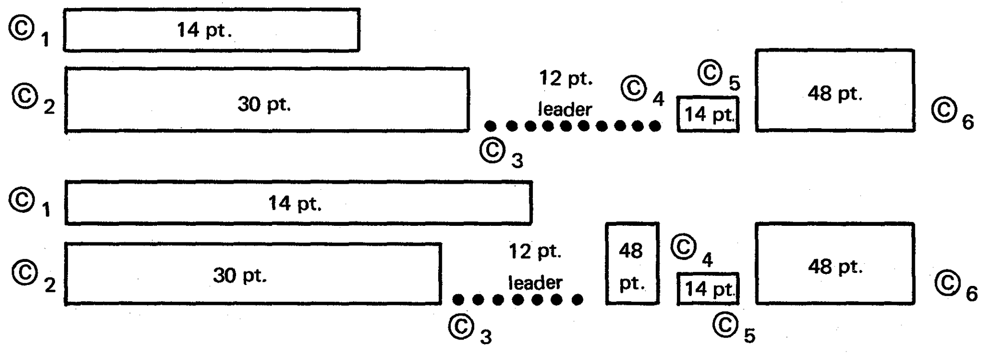
4. The leading between ads is specified in the command $cc8,24,,14,18®. This ad is set 14 points from the base line of the 8-point text to the base line of the 14-point text in the succeeding ad.

GROCERY ADS

The following two grocery store ads are set using a single format as explained below.

Texas Ruby Red
GRAPEFRUIT.................. bag 37¢

New Texas, Yellow Globe
DRY ONIONS............. 3 lbs. 19¢



Format Defined:

```
$f4d
$cc6,14,,30 Ⓡ
     Ⓒ
(QL) $cc16,30 Ⓡ
     Ⓒ
(QM) $cc14,12 Ⓡ (WL) $cc16,48 Ⓡ
     Ⓒ
$cc6,14 Ⓡ
     Ⓒ
$cc16,48,,30 Ⓡ
     Ⓒ
(QR)
     Ⓒ
     Ⓒ
```

Format Used:

```
$cw27 Ⓡ
$f4c Ⓡ
     Ⓒ
Texas Ruby Red
     Ⓒ
Grapefruit
     Ⓒ
     Ⓒ
Bag
     Ⓒ
37¢
     Ⓒ
     Ⓒ
New Texas, Yellow Globe
     Ⓒ
Dry Onions
     Ⓒ
3
     Ⓒ
Lbs. (etc.)
```

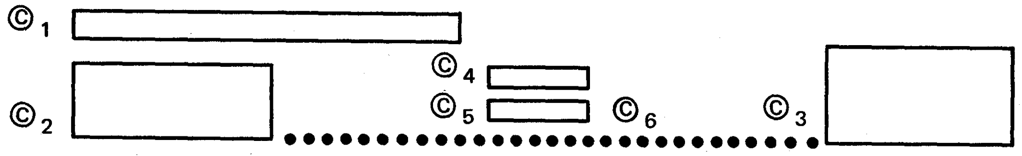
HELPFUL HINTS:

1. In this first ad, the first 48-point number is missing, but the ad can be set with the prepared format with two 48-point numbers. This is accomplished by typing the format as marked for two 48-point numbers and omitting text at the place where the 48-point number occurred.

2. The leading between the ads is specified in the $cc16,48,,30 Ⓡ command. The format is set for 30 points between ads, measured from the base line of GRAPEFRUIT; to the base line of New Texas, Yellow Globe.

GROCERY AD SET ON A PHOTON 560 WITH REVERSE LEADING
(adaptable to a PHOTON 513)

GERMAN CHOCOLATE
CAKE.......................Perfect Dessert.......................**69¢**
BORDEN OR DIXIE
PURE CREAM..........Pint Carton..........**29¢**



HELPFUL HINTS:

1. More than 128 characters are required to define this format. Consequently, an illegal format message results if the definition is prepared with all the characters. The number of characters is reduced by defining these $cc commands prior to defining the format. The commands $ce, $cf, and $cg, which replace the $cc6,12,,14 Ⓡ, the $cc14,36,,24 Ⓡ and the $cc6,10,31.3,12 Ⓡ commands, respectively, require 30 less characters.

2. The 36-point and 54-point text are saved in hold registers and used after the tab stops are established.

3. The position of the 10-point text on the second and third lines is determined by tab stops. These tab stops are established by specifying no-flash, no-leading, tab text and retrieving the 36-point and 60-point text from the hold registers for counting. This locates the midpoint for the 10-point text.

4. Flashing and leading are restored by the (QR) command and the 10-point text on lines two and three is set tab center. The tab stops are removed by the $tr Ⓡ command.

5. The 36-point text is retrieved from the hold register again and set quad middle. Point size is changed to set leaders in 14-point type. Base alignment is obtained by the use of reverse leading. The 60-point text is also retrieved from the hold register again and set quad right.

Defined Commands:

```
$cdec6,12,,14 Ⓡ
$cdfc14,36,,24 Ⓡ
$cdgc6,10,31.3,12 Ⓡ
```

Format Defined:

```
$f5d
$ce Ⓡ
          Ⓒ
(QL) $h0s
          Ⓒ
Ⓡ $h1s
          Ⓒ
Ⓡ $cf Ⓡ $cn Ⓡ $of Ⓡ $h0r Ⓡ $tt Ⓡ (QM)
$cp60 Ⓡ $tt Ⓡ $h1r Ⓡ (QR) $cg Ⓡ (TC)
          Ⓒ
(TC) (QL) (TC)
          Ⓒ
(TC) $cc16,36,,6 Ⓡ (QL) $tr Ⓡ
$h0r Ⓡ (QM) $cp14 Ⓡ (WL) $cc,60,,39 Ⓡ
$ca-17 Ⓡ $h1r Ⓡ (QR)
          Ⓒ
          Ⓒ
```

Format Used:

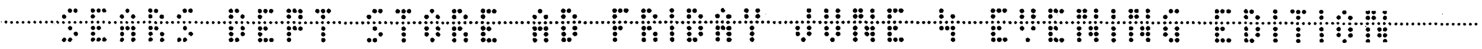
```
$cw30 Ⓡ
$f5c Ⓡ
     Ⓒ
GERMAN CHOCOLATE
     Ⓒ
CAKE
     Ⓒ
69¢
     Ⓒ
Perfect
     Ⓒ
Dessert
     Ⓒ
     Ⓒ
BORDEN OR DIXIE
     Ⓒ
     (Etc.)
```

TAG LINES

Tag lines are very helpful in the identification of output tapes; unless these tapes are processed immediately after preparation, the tag line format should be used to insert identification.

The following format definition reproduces the tag line in visual form at the beginning of the output tape.

The output tape sets, on film, a 12-point tag line on a 42 pica measure, followed by 72 points of leading. The leading prevents oversetting in the first ad.



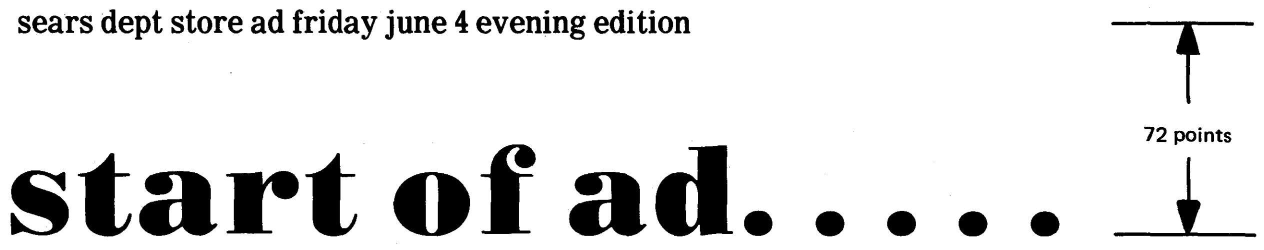
Tag Line Reproduced on Paper Tape

Format Defined:

```
$f50d
$cc 1,12,42,12®
$vt
     ©
      NOTE: If more than
®    72 pt is desired put in
      a $ca here.
     ©
     ©
```

Format Used:

```
$f50c®
          ©
sears dept store ad friday,
june 4 evening edition
          ©
```



Tag Line Reproduced on Film

APPENDIX A

Abbreviations for Special Keyboard Keys

(QL)	Quad Left
(QR)	Quad Right
(QC)	Quad Center
(QM)	Quad Middle
(TL)	Tab Left
(TR)	Tab Right
(TC)	Tab Center
(TA)	Tab Around
(WL)	With Leaders
(WX)	Leadering with other characters
(J)	Justify
(DEL)	Delete
(CALL),(C), or C	Call
(RET),(R), or R	Carriage Return

APPENDIX B

Command Groups

A. C-Group (Change Group)

			Page
1.	$CAx(RET)	Add Leading x = amount of leading in points	8

2.	$CCv,w,x,y,z(RET)	Change Combined
		v = type face
		w = point size
		x = column width
		y = leading
		z = set size
			8

3.	$CDx(command)(RET)	Change Command Define x = letter assigned to new command	32
4.	$CLx(RET)	Change Stored Leading x = amount of leading in points	7
5.	$CN(RET)	No Leading	8

6.	$CPx,y(RET)	Change Point Size and Set Size
		x = point size
		y = set size
			6

7.	$CTx(RET)	Change Type Face x = type face	5
8.	$CWx(RET)	Change Column Width x = column width in picas and points	6
9.	$CY(RET)	Change Command Dummy	35
10.	$CZx(RET)	Change Command Delete x = letter assigned to command being deleted	34

APPENDIX B (Cont)

Command Groups

			Page
B.	H-Group (Hold Group)		
1.	$HxR(RET)	Recall Text x = number assigned to text	29
2.	$HxS(text)(RET)	Save Text x = number assigned to text	28
C.	I-Group (Indent Group)		
1.	$ICx,y(RET)	Column Indent x = left indent in picas and points y = right indent in picas and points	11
2.	$IDx(command)(RET)	Indent Command Define x = letter assigned to new command	32
3.	$IHw,x,y,z(RET)	Hanging Indent w = left indent in picas and points x = right indent in picas and points y = number of lines to wait before indenting z = number of lines to indent	12
4.	$IR(RET)	Remove Column Indents	12
5.	$IS(RET)	Stop Hanging Indents	14
6.	$IY(RET)	Indent Command Dummy	35
7.	$IZx(RET)	Indent Command Delete x = letter assigned to command being deleted	34

Command Groups

			Page
D.	L-Group (Length Fit Group)		
1.	$LDx(command)(RET)	Length Fit Command Define x = letter assigned to new command	32
2.	$LLx,y(RET)	Left Length Fit x = amount of indent in picas and points y = vertical length in picas and points to which indent applies	15
3.	$LRx,y(RET)	Right Length Fit x = amount of indent in picas and points y = vertical length in picas and points to which indent applies	15
4.	$LS(RET)	Stop Length Fit Indents	16
5.	$LY(RET)	Length Fit Command Dummy	35
6.	$LZx(RET)	Length Fit Command Delete x = letter assigned to command being deleted	34
E.	O-Group (Output Format Group)		
1.	$OC(RET)	Ragged Center Format	23
2.	$ODx(command)(RET)	Output Command Define x = letter assigned to new command	32
3.	$OF(RET)	Omit Flash	27
4.	$OHx(RET)	Output Hyphenable x = word spacing at which hyphenation should be attempted	24
5.	$OI(RET)	Inhibit Logical Hyphenation Rules	25

		Command Groups	Page
6.	$OJx(RET)	Minimum Justifying Space x = minimum word space (in relative units)	26
7.	$OL(RET)	Ragged Left Format	23
8.	$OMx(RET)	Ragged Middle Format x = amount of stagger (in points)	23
9.	$ON(RET)	Output Non-Hyphenable	25
10.	$OR(RET)	Ragged Right Format	23
11.	$OSx,y(RET)	Use Letter Spacing x = word spacing at which letter-spacing should be attempted y = maximum letter-spacing which can be used	26
12.	$OU(RET)	Enable Logical Hyphenation Rules	25
13.	$OW(RET)	Inhibit Letter Spacing	26
14.	$OY(RET)	Output Command Dummy	35
15.	$OZx(RET)	Output Command Delete x = letter assigned to command being deleted	34
F.	S-Group (Skew Group)		
1.	$SDx(command)(RET)	Skew Command Define x = letter assigned to new command	32
2.	$SLx,y(RET)	Skew Left x = horizontal length of triangle in picas and points y = vertical length of triangle in picas and points	16

Command Groups

		Page
3. $SRx,y(RET)	Skew Right x = horizontal length of triangle in picas and points y = vertical length of triangle in picas and points	16
4. $SS(RET)	Stop Skews	18
5. $SY(RET)	Skew Command Dummy	35
6. $SZx(RET)	Skew Command Delete x = letter assigned to command being deleted	34
G. T-Group (Tabular Group)		
1. $TCx(RET)	Compute Equal Tab Stops x = number of sub-columns	19
2. $TDx(command)(RET)	Tab Command Define x = letter assigned to new command	32
3. $TPx,y,...,z(RET)	Tab Proportionate x,y,...,z = proportions into which column is divided	20
4. $TR(RET)	Remove Tab Stops	22
5. $TSx,y,...,z(RET)	Tab Set x,y,...,z = distances from left margin (in picas and points) at which tab stops are set	19
6. $TT(RET)	Tab Text	21
7. $TY(RET)	Tab Command Dummy	35

Command Groups

			Page
8.	$TZx(RET)	Tab Command Delete x = letter assigned to command being deleted	34
H.	V-Group (Visual Group)		
1.	$VDx(command)(RET)	Visual Command Define x = letter assigned to new command	32
2.	$VP(message)(RET)	Visual Punch	30
3.	$VT(message)(RET)	Visual and Tag Line	30
4.	$VY(RET)	Visual Command Dummy	35
5.	$VZx(RET)	Visual Command Delete x = letter assigned to command being deleted	34

APPENDIX C

Errors

The PDP-8 Photon 513-1560 System is dependent upon the accuracy of the mark-up man and the perforator operator for successful operation. The system can only do exactly what it is told to do; errors in the text are not detected by the system. The text will be printed exactly as it is punched.

Many other errors, however, are detected by the system. Whenever an error is found, a message is punched on the output tape, readable to the human eye, describing the error. These messages are punched as the errors are encountered, so they may appear anywhere in the output tape (not just at the beginning or end of the tape). Most of the messages are self-explanatory.

It was noted in Section 2.1 that type face, point size, column width, and leading must be specified at the beginning of each input tape. If any of these parameters are omitted, the appropriate message will be punched, and none of the input will be processed. The message will tell which parameters were omitted; for example, the message might be COLUMN WIDTH IS UNDEFINED, or it might be LEADING AND POINT SIZE ARE UNDEFINED. As a precaution, some tape-feed should always be punched at the beginning of an input tape so that any missing parameters can be added without re-punching the whole tape.

Many errors occur because of incorrect use of commands. The computer will understand commands only if they are punched in the exact form given in this manual. Any extra punctuation, missing punctuation, etc., will cause an error. The message punched will usually identify which group the incorrect command belongs to and what is wrong with the command. The message REDEFINITION ERROR means that a new command was defined using a letter belonging to one of the basic commands in that group. (The new command will not be defined.) The message ILLEGAL DELETION means that a deleting command was used to delete one of the basic commands in that group. (The basic command will not be deleted.) The message UNDEFINED OR ILLEGAL COMMAND means that either a command which does not exist was called for or there was an error in the format of the command (wrong punctuation, missing numbers, etc.). The message EXCESSIVE RECURSION means that a newly-defined command was defined in terms of another newly-defined command, which in turn had been defined in terms of a newly-defined command, and so on; and the chain of definitions is too long (more than seven).

There are no input errors which cause the computer to stop. If the computer stops without punching a message, it is due to a malfunction of the equipment.

When an error occurs in a command, the computer punches the appropriate message, ignores the whole command, and keeps going. The printed output, of course, will not have the desired form, but it may only be necessary to re-process small parts of the text. As a precaution, some tape-feed should be punched before and after all commands so that they may be corrected if necessary without re-punching the entire tape.

One of the most troublesome errors is the omitting of a (RET) at the end of a command. When this happens, not only is the command ignored, but the next command or some of the following text may also be lost.

Another pitfall is the use of the \$. If a \$ is to be printed as part of the text, two \$'s must be punched. If the computer encounters a single \$, it assumes the \$ is the beginning of a command and will process only whatever follows as a command.

Much time and re-processing can be saved on key-punch mistakes if the perforator operator notices the error soon enough. Errors in both text and commands can be corrected by use of the delete key (see Section 2.10.5).