LIGHTNING BASIC EXTENSION

INSTRUCTION MANUAL

CP SOFTWARE

LIGHTNING BASIC EXTENSION

Mallard was a steam engine which still holds the speed record, 126 m.p.h. Mallard, the engine, was fast, powerful and efficient. Mallard, the BASIC which is supplied with your PCW is also fast, powerful and efficient. Yet Mallard, the engine, was clumsy at doing routine jobs like shunting, and there were many lines available on which it was not allowed to run. There are also a number of interesting things that the PCW as a computer CAN do, but Mallard CANNOT access. (You'll be surprised at what the PCW can in fact do.)

LIGHTNING is designed to overcome these deficiencies. It works in harness with Mallard and allows you a great number of new commands which can be entered just as simply as the standard Mallard BASIC commands. It is not simply an extended BASIC, but actually three extended BASICs that can work alone or in partnership.

There is a little trick that we have to use. We take just one word from the Mallard vocabulary, and allow it to divert the BASIC to our new commands. That word is LET. (You don't use it anyway,- unless you enjoy doing totally unproductive typing!) All the other Mallard words work normally, although in one or two cases their power has been extended.

This means that we have to make up a new word, which really has to be a three letter one starting with L. Why not LEB? This stands for LIGHTNING EXTENDED BASIC! Some of the routines were developed from CP software's "All you ever wanted to know about Graphics, the Universe and Everything on the PCW". (We'll call it AYE from now on!) Others were taken from the author's LNER and SPREX programs. Others are totally new. LIGHTNING is NOT intended to replace AYE which is the ideal medium for those wishing to program in say C, Pascal or Assembly code.

FIRST OF ALL, LETS TRY THE DEMO

Put your system master disc side 2 into the drive, and you will see the normal CP/M prompt. Type in: BASIC (and press RETURN). This will take you into Mallard BASIC. Now put in the LIGHTNING disc and type: run"demo" (and press RETURN). You will be asked to press A or P, so press P for the moment. (Pressing A gives an AUTORUN but P allows you to inspect pages of the DEMO and press a key when you are ready to continue.)

At any time in the DEMO, you can press STOP to exit from it. When you see the DODECAGRAM again, you will have run all the pages of the demo, but it will go on running round again if you wish. Some pages may be slightly different each time round. When you exit from the program you will actually have all of LIGHTNING loaded, but we suggest that you switch off the computer and then follow the instructions in the next section for a normal load of LIGHT1.

When you start to use this program, we would advise starting with just LIGHT1. This doesn't take a great deal of memory, but it provides what probably 95% of the users will want 95% of the time. Enough of the waffle, let's get down to it!

LOADING.

To load LIGHTNING, first put a copy of Master disc 2 into the computer to go into CP/M. Then type BASIC <> and you will see the usual Mallard message appear. (We shall use <> to mean press the enter or return key). Now put the LIGHTNING disc into the drive and type run"light"<>. You will get a prompt: Edit keys? Y/N. For the moment press N and you will be in the new BASIC.

Later you will probably find it easier to include BASIC on a copy of the LIGHTNING disc, (and maybe CP/M also). If you do so, you can go from CP/M to this stage by simply typing: BASIC light<> You could, of course, use a PROFILE.SUB to get there from switch on.

You can, at this point, load any BASIC program you have. Provided you haven't used any LETs, it will work normally. However, you will probably prefer to try out the new features of LIGHTNING, so we will look at the syntax, before discussing individual commands.

SYNTAX.

Let us take one simple command, namely a screen clear, and see the various ways that LIGHT can deal with this. The full command is: LEB cls<> If you type this in, you will see it work. However all LEB commands have abbreviated forms and once you get used to them, you will almost certainly use them and save your typing fingers! LEB commands can be replaced by the # symbol (next to the return key) in the same way that PRINT statements can be replaced by ?. If you use the #, you should NOT leave a space, (although it won't matter if you do).

As well as this facility, words like cls can be shortened to either one or two letters. In the case of cls, you can replace it by the single letter c. The interpreter doesn't mind whether you use capitals or small letters. Thus, the four conventional ways of entering our command are:

LEB cls<>
LEB c<>
#cls<>
#c<>

There may be other ways that the computer would accept. The interpreter tries to think what you mean, but it is best to stick to the suggested ones. If you use the # method in a program line, the computer understands, but in a subsequent listing, it will appear in LEB form. This is just the same as the case of using ? for PRINT. When we introduce each command we shall give it in full as LEB cls, and also its briefest form #c. If it is followed by other abbreviations in brackets, these are other commands that are connected.

PARAMETERS.

PLEASE READ THIS CAREFULLY, IT IS RATHER IMPORTANT!
Most commands, whether in Mallard or LIGHT, need additional pieces of information. We shall refer to them as PARAMETERS if they are numbers,

and ARGUMENTS if they are filenames, words or string variables. Arguments use various types of syntax, and we will deal with this when we come to them. However, most of our new commands just need parameters.

Again let us see how it works with an example. Suppose that you wish to print something on the screen at a certain point. Use LEB at (or #a). Obviously you have to tell the computer just where you want it to print. The command will need two parameters, the row number and the column number. To print "MYPROG" at the centre of the screen, you would need to start the print on row 16 at column 42. So we enter this line:

```
10 #a,16,42:?"MYPROG"
In the listing this will appear as
10 LEB a,16,42:PRINT"MYPROG"
```

There is an important point to notice here. Not only must there be a comma between the 16 and the 42, but there must be a comma BETWEEN THE a AND THE 16. This syntax is essential whenever you use an LEB command that needs parameters.

This command clearly needs just 2 parameters. Unlike Mallard, if you put the wrong number of parameters you won't get an error, - the computer will just do what it thinks you meant it to do! If you put too many the excess will just be ignored and the first two taken. If you put too few the computer will use what is already in the parameter buffer. It will not crash, but the result might be peculiar!

All LEB parameters are whole numbers between 0 and 65535. We shall use the convention that a single letter parameter means a number between 0 and 255, and a double letter means a number 0- 65535. For instance if POKE was a non-Mallard word we would describe it by POKE aa,b. An optional parameter will be put in brackets. Sometimes a parameter can take only a few values, say 0-7. In this case if the value is too high, the computer replaces it by the value it thinks you meant. We shall reserve the letters r and c for rows and columns, and xx and y for co-ordinates.

Just a little point here, that we shall have to make at some stage. Mallard and CP/M use the convention that rows come before columns. We shall use that here. However, for graphical displays, many people think in terms of co-ordinates. Where this applies xx will be measured across the screen, and y from 0 at the top down the screen.

One other point about parameters. They can be entered as variable expressions. If a parameter evaluates to a decimal, it will simply be rounded to a whole number. There will be an error if the evaluation is less than -32768 or greater than 65535. (That shouldn't often happen!)

LIGHT 1 COMMANDS.

LIGHT (or LIGHT1) contains the workhorse of LIGHTNING and the commands that you are most likely to use frequently. Commands that need care are

omitted, and none of the words in LIGHT 1 should crash the machine into more than a simple BASIC error statement however carelessly you use them! There is one exception. #doke is a double poke, and you will probably be aware that POKEing carelessly can have strange (though thankfully not permanent) effects on the PCW's health! LIGHT 1 routines are generally fairly short, and you will probably be surprised at how much we have managed to pack into just under 2.5K, so that you will have the use of nearly all the memory usually available. Here are your new commands:

LEB at,r,c #a (#h,#w,#cs,#cr)
Places cursor at row r, column c for subsequent PRINT statements.

LEB beep,(a) #b (#y)
This is really three different commands in one. If given a parameter 1-254, the computer BEEPs that number of times. #b,255 turns the beeper on permanently... well, no, not permanently since thankfully you can turn it off with #b,0 (#b without a parameter gives a single beep).

LEB cls #c (#h)
Clears the screen. You won't have to worry about this at the moment, but actually it is more than a simple clear screen in that it sets Roller Ram back to its initial state. (PRINT chr$(27)"H"chr$(27)"E" does not). This allows some of the routines in LIGHT 2 and LIGHT 3 to work more efficiently and quickly.

LEB condensed #co (#lt,#lr)
Printer to print in condensed style.

LEB cstore #cs (#cr)
Stores the current position of the cursor until restored.

LEB crestore #cr (#cs)
Restores the previously stored cursor position.

LEB disable #d (#e)
Turns the cursor off - invisible might be a more correct description.

LEB deek,aa #de (#do)
Peeks the bytes at aa and aa+1, calculates 256 * the second + the first and prints the result.

LEB doke,aa,bb.
Turns bb into two bytes, and double pokes it into the locations aa and aa+1. If you are unfamiliar it may be helpful to give a BASIC line which does the same: h=int(bb/256):l=bb-h*256:poke aa,l:poke aa+1,h #de, and #do will be mainly useful for those who are dabbling in assembly or machine code.

LEB dump,aa,b #du
Dumps b bytes from memory starting at aa onto the screen. Mainly useful for code programmers, but a useful aid to beginners who want to see how the PCW stores its code, and quite safe to use. There are 10 bytes per line.

LEB enable #e (#d)
Turns the cursor back on, after disabling with #d.

LEB fill,xx,y #f (#l,#re,#rf)
Fills the area surrounding the point xx,y. This should be fully tested if you are using it in a program as more than one fill may be required, or a careful choice of co-ordinates may be needed (e.g in a triangle). The principle is that it fills a line from the point xx,y, and then goes up and down and repeats this until an obstruction is met. It isn't difficult to use, it just needs care.

LEB flash,(n) #fl (#i,#o)
Flashes the screen into inverse and back n times.

LEB getkey #g (#xg)
Changes the operation of INKEY$, so that it waits for a key to be pressed before continuing the program. Saves all those standard routines that we've all written! It is reversed by #xg.

LEB home #h (#c,#w)
Sends the cursor to the top of the screen without clearing it. It is also used to cancel a window created by #w.

LEB invert #i (#o,#fl)
Inverts the screen colours. #o turns it back.

LEB k,n:"xxxxx" #k
This is a function key setter, which hasn't previously appeared in any PCW BASIC as far as we know. The syntax should be studied carefully. The parameter n should normally be between 1 and 8, and it will apply to the function keys f1-f8. Parameters between 0 and 25 CAN be used, but with care as other keys (like the cursor keys) may be set and lose their normal operations. (Study the manual if you want to do this).

The string "xxxxx" can contain any characters you like. There is no precise limitation on length, but if the TOTAL length of the key settings is too great #k will return "Improper Argument". Clearly, it is desirable that a function key may contain a RETURN so that it operates immediately and also a " mark. If you need these but ! at the end for a RETURN and use ' instead of ". The string DOES need to be closed at the end by ", so this is why the adjustment is needed.

LEB linedraw,xx,y,pp,q #l (#xl,#p,#re,#f)
This is a high resolution command. The screen goes from 0-719 (left to right) for the x co-ordinate, and from 0-255 (top to bottom) for the y co-ordinate. The command plots a line starting at (xx,y) and going to (pp,q).

LEB lecho #le (#n)
Echoes all PRINT and DISPLAY commands to the printer. Note that TYPE still only goes to the screen.

LEB locate,aa #lo
Locates and prints all occurrences of the two byte number aa in memory. The command itself will cause some occurences but the relevant ones will soon be obvious.

LEB lreset #lr (#co,#lt)
Resets the printer to normal after #lt, #co or other control sequences.

LEB ltext #lt (#co,#lr)
Sets printer to LLIST exactly as the LISTing would appear on the screen. The type is condensed.

LEB message #m (#xm)
If the status line is turned off by #xm, this will regain the message. It is worth noting that if graphics go over into the bottom line of the screen with the status message on, a #c will not clear them. The trick to use is #m.

LEB noecho #n (#le)
Turns the printer echo off.

LEB off #o (#i,#r,#u)
Turns off inverse screen, reverse video printing mode or underline mode or any combination of the three at the time.

LEB plot,xx,y #p (#xp,#l)
Plots a single pixel on screen at xx,y. (See #l for co- ordinates). Note that xx should NOT exceed 719. If it does it will plot on the next down, and in extreme cases right off the screen, possibly damaging the character set or roller ram. You can try if you want, but don't blame us for a crash!

LEB page #pa
A single routine that does away with the all the hassle of "Press any key to continue", etc, etc. Try it and see!

LEB rvideo #r (#o)
Turns on a mode where all PRINT commands will operate in reverse video.

LEB rect,xx,y,pp,q #re (#l,#rf)
Draws a rectangle. (xx,y) and (pp,q) must be OPPOSITE corners.

LEB rfill,xx,y,pp,q. #rf (#re)
Draws a rectangle as above and fills it. If the corners are NOT the top left and bottom right, you are liable to get an unexpected result! This command may not work with very small rectangles. This command should not be used with medium resolution (see LIGHT 2).

LEB search:"xxxxx": or LEB search:yyyyy: #s
This very useful debugging tool will search through a BASIC program and report all line numbers where the argument occurs. QUOTE MARKS should be used if

you expect to find the argument in a PRINT, DATA or REM statement, or as part of a filename. If in doubt try with quotes and then without quotes. The BASIC program:

```
10 GOSUB 30
20 END
30 PRINT"MANY HAPPY RETURNS"
40 RETURN
```

would give 30 from #s:"RETURN": and 40 from #s:RETURN: The search MAY not find such arguments as GOTO 1000, GOSUB or v-78 after a program has been run as Mallard sometimes changes the coding while the program runs. (EDITing a program will restore the normal coding). Colons MUST be used at the start and finish of the argument. Details that are usually unimportant (extra spaces, capital letters) can affect a search.

LEB time #t (#ts)
Tells the time in hours-mins-secs on the internal computer clock.

LEB tset,h,m,s #ts (#t)
Sets the internal clock to the time in the parameters. It may be that you wish to get the time as a variable. If so define a function by: DEF FNT(n)-PEEK(n)-INT(PEEK(n)/16)*6 then for hours, mins, secs: h-FNT(64502):m-FNT(64503):s-FNT(64504)

LEB tabchar,n #ta
TAB will normally tabulate with spaces. If this function is used TABbing will be with the character given by the ASCII code n. For example to TAB with ..... #ta,46. #ta,32 reverts to normal TAB.

LEB underline #u (#o)
All subsequent PRINT commands will be underlined. #o turns this off.

LEB vlist #v
Lists all the variables with their values that are in memory (not array variables). String variables containing CHR$(27) will have it printed as CHR$(255). Otherwise you might clear the screen in the middle, etc.!

LEB window,r,c,a,b #w (#h,#re,#r)
Creates a window top left corner at row r, column c, a rows deep, b columns wide. All subsequent PRINTing takes place in this window, so don't make it too small! Suitable use of #re and #r can make it look like a pull down menu. To cancel, either create a new window or use #h.

LEB waitsecs,n #wa
Waits for n seconds. That one's simple!

LEB xgetkey #xg (#g)
Reverts INKEY$ to usual operation.
LEB xlinedraw,xx,y,pp,q #xl (#l)

Unplots a line drawn with #l and the same parameters.

LEB xmessage #xm (#m)
Turns off the status line. All 32 screen lines can then be used.

LEB xplot,xx,y #xp (#xl,#p)
Unplots a pixel plotted at (xx,y).

LEB yodel,(n) #y (#b)
Yodels n times. A 'yodel' is a two-tone beep. The routines in this command can be used to make music (???!) of your own. See TECHNICAL POINTS if you want to try. Rather you than me!

LIGHT1 also has one important function and one or two enhancements of the original Mallard interpreter.

EXP(a$)
Unlike VAL, EXP will attempt to evaluate the string as if it was a mathematical expression. An example will show. If you enter:

a$="5*sqr(49)":?val(a$),exp(a$) you will get: 5 35

Our GRAPH program will show its power. EXP used with a NUMERICAL argument works as in Mallard. **WARNING**: EXP uses the buffer of the highest file available so you should not have an open file in buffer 3 while using it.

POKE nn,a,b,c,.....z
You can now use a multiple POKE, a feature of many recent BASICs but not allowed in Mallard. POKEs in a,b,c... into successive memory locations starting at nn. Makes DATA statements unnecessary in writing code routines, and should be useful in creating SPRITES (see LIGHT 3).

LIST
This can now be used as a program line. The coding actually uses not a single byte more, and is particularly helpful in demo or educational programs.

ERRORS
Mallard produces the line to be edited when it comes across a SYNTAX error. LIGHT produces it for most errors. On this subject. LIGHT uses mainly SYNTAX or IMPROPER ARGUMENT errors as appropriate, but if you come across UNKNOWN USER FUNCTION, it probably means that there is an LEB function of that name, but it is in a section that you haven't loaded yet. See also LEB user in LIGHT 2.

END
If the cursor is disabled by your program, either an END statement, a break or an error will reenable the cursor.

LIGHT 2 AND LIGHT 3 (LOADING INSTRUCTIONS)

If you think of LIGHTNING as a 3-speed gearbox, it is all very easy. To change up a gear from LIGHT 1 to LIGHT 2, just run"ext2"<>, and from LIGHT 2 to LIGHT 3 run"ext3"<>. You can't go from gear 1 to gear 3. Alternatively you can start in a higher gear. Instead of BASIC LIGHT<> you can load with BASIC light2<> or BASIC light3<>. Changing down a gear or two is possible by using the memory command though if you do this, be careful not to use commands in your erased section.

SPRITES AND ICONS.

LIGHTNING BASIC offers you the facility to use Sprites or Icons (or both). Each have advantages and disadvantages:

The arguments for ICONS are: 1. They take far less memory. 2. They don't destroy what they tread on. 3. They can be used more than once on the same screen.

The arguments for SPRITES are: 1. They can be placed at any pixel on the screen rather than starting at the top of a character square. 2. They can have a much larger or more varied size. 3. They can use the frame flyback technique to make operation smoother.

ICONS are in LIGHT 2, SPRITES in LIGHT 3, so let's deal with ICONS first, starting with user defined graphics (UDGs).

LEB udg,n,a,b,c,d,e,f,g,h #ud (Icon commands, #st)
The first parameter names the character to be used. If n=178 say, the result of PRINT chr$(178) will be that YOUR design of this character appears on the screen. It is sensible to restrict n to values between 128 and 255, as this will not alter characters in normal use. There is, of course, nothing to stop you redesigning A by taking n=65. The other 8 parameters give each row of the character as a number developed from binary notation. Too difficult? Well, our program ICONS will overcome any problems.

All Icons are squares 16 pixels across and 16 pixels down. An icon is created from 4 successive characters, of which the first only is named.

LEB cicon,m,n #ci
Creates ICON number m. m=0 to 7. (If you use 8 instead of 0, nobody will notice)! The icon consists of the 4 UDGs starting at n. Example: #ci,2,65 Icon number 2 would look like this:

AB
CD

At this stage it is probably helpful to think what the computer will have to do to operate the icons. These routines are needed:

1. IDENTIFY which icon is being used.

2. STORE what is on the screen before the icon is placed there.
3. PLACE the icon on the screen.
4. REMOVE the icon from the screen and replace what was there before.

LEB **nicon,m #ni**
Before putting an icon on the screen, you must create it (#ci), and later name it (#ni). Icon commands following this refer to icon m until another #ni is encountered. This routine calls IDENTIFY.

LEB **picon,r,c #pi**
The first appearance of an icon on the screen should be done by #pi (not by #mi, as this might do odd things on the screen, - try it if you have a sense of humour)! The co-ordinates are the top left of the icon, and the r should be 0-30, and the c should be 0-88. An error may be given if you get this wrong. This needs STORE and PLACE.

LEB **micon,r,c #mi**
Moves the current icon from its current position to row r, column c. It also replaces whatever was on the screen at the old position. This is done by REMOVE, STORE and then PLACE.

LEB **xicon #xi**
Turns off the current icon. Note that you can have the same icon on the screen more than once if you use two (or more) #pi commands on the same icon without a #xi in between, but the old copy will remain on screen until the next #c. This routine just needs REMOVE.

OTHER LIGHT 2 COMMANDS

LEB **address:word: #ad**
Gives the address of the routine associated with the reserved word in Mallard, provided it is in the main command or main function table. Mainly for hackers, as is....

LEB **transfer,aa,bb,cc #tr** Transfers memory FROM aa TO bb. The number of bytes is cc. Needless to say, be careful with this one and don't blame us if you crash!

LEB **gmedium #gm (#gh)** You may think it is odd, but many graphics screens look much better in 360*256 resolution rather than the full 720*256. If you think of the shape of the screen, you will see why this is. #gm affects several commands and having invoked it, you think of a screen line as 360 bytes wide, so that the xx in the parameter goes from 0-359 not 0-719. The routines affected automatically by #gm are:
#l,#xl,#p,#xp,#e.
It is not too easy to use #f or #rf in medium resolution.

LEB **ghigh #gh (#gm)**
This cancels #gm and you are back to 720*256 resolution in the routines mentioned in the above paragraph.

LEB **sload,n #sl (#ss,#ds)**
Loads a saved screen from #ss on to the screen. See below.....

LEB **ssave,n #ss (#sl,#dl)**
This command saves the current screen on to the memory disc. n=1 or 2 and two such screens can be held concurrently. This makes for a very quick swapping of screens, and the main use may be to create a HELP screen by #ss,1 and use area 2 to save the current screen e.g:

```
300 #g:if upper$(inkey$)="H" then #ss,2:#sl,1:#pa:#sl,2:#xg
```

BUT: IF YOU USE THIS FACILITY YOU PROBABLY DESTROY ANYTHING ON THE M DISC IF YOU HAVE NOT LEFT AT LEAST 70K SPARE. After using #ss or #sl it is probably sensible to use DIR before you make a normal SAVE. (This is not a bad idea even in normal Mallard after a disc change, since when you change discs the old directory can remain in memory causing problems). #sl and #ss will not operate if the computer thinks there is not sufficient room on M Disc when EXT2 is loaded.

LEB **sdump #sd**
This gives a full screen dump that fills one side of an A4 sheet of paper. It is very quick in comparison to the dump that you can get with GSX. (Ever tried it? No? I can't say I blame you!) There are two things to watch. Firstly you should NOT have scrolled the screen since your last #c. (#sd assumes Roller Ram initialised and so is quicker and more economical in memory). Secondly, it does temporarily annex about 780 bytes of memory, so you should not use it on any program that stretches memory to the maximum, or you may lose the end of your program, or destroy your string variables.

LEB **gprint,n:s$: #gp (#st)**
This sends to the printer a string of characters (including UDGs) and prints them in hard copy. s$ or any other string variable is a normal BASIC string, which must have been defined before the command is used. If you use your own UDGs, put them in s$ by using chr$. For instance if you have a drawing of a cat in characters 180 and 181,- s$="CAT "+chr$(180)+chr$(181) can be used. The string variable should contain less than 80 characters. The parameter n should contain 0, 1 or 2 for normal size, double width or double width and height characters respectively.

LEB **stringprint,(n):s$: #st (#gp)**
This works very much like #gp (see above) except that this time the printing goes to the screen instead of the printer. There is an additional facility in that the type of printing, 0, 1 or 2 (which mean the same as above) may be included in the string, so that you can change horses in midstream for example:

```
s$=chr$(0)+"cat"+chr$(1)+"tiger"+chr$(2)+"elephant"
```

This is why the parameter n is optional.

LEB mload,aa,bb:"xxx.yyy": #ml (#ms)
LEB msave,aa,bb:"xxx.yyy": #ms (#ml)
These two commands enable you to save and load a section of memory to disc. aa is the start address of the section and bb is the end address. The "xxx.yyy" can be any characters subject to the normal restrictions on file names and extensions. **WARNINGS:** These commands (as do #dl and #ds) use CP/M routines, and so must be handled with care. Error checking in CP/M is present, but if an error is detected you will lose your BASIC. A particular one to watch is that you DO NOT try to save a file with the same name as one already on disc. If a load is attempted on a non-existent file, the command will be ignored and you won't get an error message. The routine loads and saves to the default drive, so use OPTION FILES if you want to change drives. It is probably advisable to use DIR after using one of these commands before doing a normal SAVE. The save is taken 128 bytes at a time, so when you reload with #ml, make sure that the load doesn't wipe out code that you want to keep. As an example, #ml,51200,51207:"my.fil": would affect anything that was up to 51327 in memory. You don't have to load the code in the same place as you saved it from.

LEB user,n,(a,b,c..) #us (See technical section.)

LIGHT 3

Now you can try the big SPRITES. There is bit pattern data in the computer for two sprites but if you want to design your own, you'll need to do it yourself. A sprite can be up to 6 BYTES wide and 32 PIXELS high. Suppose you want to design one 4 bytes wide and 11 pixels high. You have to POKE in the data, (luckily LIGHT gives you a multiple POKE). You start: POKE 55789,4,11,a,b,c.....until you have filled in the 44 bits of data starting at a. Then for another sprite, you start POKEing again at 55835 (55835=55789+2+4*11) and so on.

Conventionally, Sprites are designed on graph paper and mapped to a binary data bit pattern, as in the diamond shape below. Note that you will need at least 6 clear bits to the right of the pattern (and any other pattern you may design).

```
00000010,00000000
00000 111,00000000
0000 1111,10000000
000 11111,11 000000
0000 1111,10000000
00000 111,00000000
00000010,00000000
```

The first two data bytes are width (in bytes) and height (in pixels). The bit pattern data is in binary form and will need conversion to decimal. The data stream to

POKE is:

```
2,7
2,0,7,0,15,128,31,192,15,128,7,0,2,0        REM width, height
                                            REM binary data
```

The data buffer for storing this sprite data can contain a maximum of 388 bytes, including the two bytes needed for the width and height of each sprite.

LEB lsprite,n #ls (#ps
After POKEing in the sprite data and before plotting the sprite you must initialise using this function. n is either 1 or 2 depending on how many sprites are in the buffer at 55789.

LEB psprite,n,xx,y #ps (#ls)
Places the sprite (n=1 or 2) with its top corner at xx,y. Note that these coordinates and the length of your POKEs into the buffer are not error checked, and must be compatible with the size of the screen and the sprite maximum.

LEB xsprite,n #xs
Erases the sprite (n=1-8) from its last position.

LEB juggle,(2) #j #j
toggles the normal character set with a set of manuscript charcters. #j,2 toggles normal with a set of '2001 style' characters. Note that if you are in manuscript or in 2001 you cannot get 2001 or manuscript respectively without toggling back to normal. If you disobey this rule, you lose your normal character set (until you switch off).

LEB dload:"xxx.yyy": #dl (#ml,#ms,#ds)
LEB dsave:"xxx.yyy": #ds (#ml,#ms,#ds)
These commands load and save the screen to the disc drive of your choice. (No danger of corruption as there is with #sl and #ss). Syntax (apart from no parameters) and warnings are the same as for #ml and #ms, so read these first! Such a disc save is expensive on space (25K) and you may prefer the semi-BASIC program SCSAVE included on your disc which is usually much more economical. You can save or load with a screen that has scrolled since the last #c, but the loading looks a little odd until it all comes right in the end!

TECHNICAL POINTS

LIGHTNING is designed so that the user can add some extra commands with the minimum of difficulty. There are 8 USER functions allowed, and with LIGHT 2 loaded, there is space between 60296 and 60400 to enter the code. The address of the start of the code must be put in a table starting at 59880, two bytes for each user command going from 0-7. So to start USER 3 at 60350, you would need #do,59886,60350.

This is not something that most people will attempt, so I have used USER 2 to 7 myself. **#us,4,n** will scroll the screen by n pixels. (**#us,4,0** should restore the screen back to normal position). **#us,5,a,b** prints out a+256*b as a single

number, (sometimes useful after a dump). **#us,7** blanks the screen. **#us,6** turns it on again. These four routines are as simple as you can get in machine code, and you will see the code at 60296 to 60328 if you wish. **#us,3** de-protects a BASIC program. **#us,2** waits for frame flyback (but only if Light3 is loaded), this one helps to make for smoother sprite movement.

When you write code for a USER, you don't have to worry about preserving the registers to get back into BASIC. The 'engine room' has already arranged the stack so that a simple RET takes you back to BASIC. Any parameters you use will be in the parameter buffer at 62873 for the first, 62875 for the second etc.

Another technical area that in which some of you may want to experiment is music making. Our #y command shows that notes are possible, and while I don't think it is likely that you can reproduce Beethoven's 9th on the PCW, you may be able to do better than LIGHT in this field. Using #do with DOKEs of over 256 at the locations 61357 and 61360 are perfectly safe (except for your neighbours' sleep), and then a CALL to 61355 should produce a note. (music=61355:call music). To get a new two tone yodel, you will have to DOKE at these locations and also at 61395 and 61398.

EDITKEYS.
While we wouldn't pretend that the PCW is the easiest for BASIC editing, suitable use of AUTO, RENUM, DELETE, cursor left and cut makes it much simpler than it first appears to be. EDITKEYS is designed to make a further improvement. You can install it by saying Y to the prompt when you load LIGHT or simply run"editkeys" later. It sets the following keys:

LINE puts the cursor at the start of your editing line
EOL puts it at the end.
RELAY jumps from the start of one statement to the next in a multistatement line.
CAN clears the screen.
f1 lists the section of the program you have chosen for LISTing.
f8 saves the program.
The PASTE key pauses a listing and may be easier to find quickly than f5 which does the same.

You cannot use RPED if you change f1, f3, f5 (unless you know how to unprotect and amend it).

OTHER FILES ON DISC

Rather than give you just demonstration programs, we have included some useful programs on your disc that run with LIGHT:

AYE 1 to AYE 5
The AYE programs 1-5 were used to demonstrate AYE. They work equally well with LIGHTNING, and the coding is roughly 50% shorter . AYE2 needs at least LIGHT2. AYE 3-5 need LIGHT3.

ICONS
I don't think you will have seen anything like this one before! It is a program to create UDGs and ICONS. Instructions are included. When the program ends

you will have a different program which you can save, and merge later into your own programs so that you have ready made UDGs and ICONS.

BOXER
Just an ICON demo. It was written using ICONS to create them, and a listing will show you how the ICONS system works. ICONS and BOXER will need at least LIGHT2 loaded.

POLYGRAMS
Works from LIGHT. Pretty pictures and patterns.

SCSAVE
Works from LIGHT. Saves screens economically to disc. As this works better in medium resolution it is probably advisable to have LIGHT2 loaded. Instructions in the program.

PIBAR and GRAPH
Both work from LIGHT, but need LIGHT2 for full screen dumps and memory screensave, and LIGHT3 for a screensave to drive A or B.

PIBAR creates files which can then be reproduced in various ways as bar charts or pie charts on screen or to hard copy. Little instruction needed, but sufficient is in the program.

GRAPH plots any (yes any, well more or less any) algebraic graph you choose to enter. A full detailed set of instructions is in the Notes option.

ITALSUB
An italics character set which can be used as a subroutine as an alternative to the manuscript set. The program explains all!

CIRCSUB
A subroutine to draw circles quickly. Once again, find out how from the program.

YOURPIC
Draw your own pictures. Instructions in the program.

APETRAP
No comment, except we hope you don't fall into it!

... LATE EXTRA

I've just had a thought. you would like to know how much space you have left on the disc drive without going back to CP/M, finding DIR.COM etc.,etc., wouldn't you? And LEB q didn't do anything, did it? Well, now it does!

LEB question #q
tells you how many K you have left on the drive you are using.

INDEX

The LIGHTNING commands are listed above. Each is described in more detail in the section on LIGHT1, 2 or 3 respectively.

7900

Published by:

CP Software, Stonefield, 198 The Hill, Burford, Oxfordshire OX8 4HX

If you should have any difficulty with loading this program, please return it direct to CP SOFTWARE for our immediate attention.

For technical advice on the program, please contact either CP SOFTWARE technical dept. or the author, G. Childs, Winchcombe, Clos.