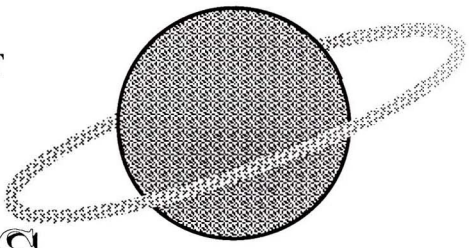
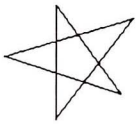
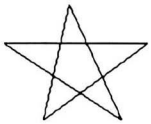
ALL YOU EVER
WANTED TO KNOW ABOUT



GRAPHICS, THE UNIVERSE AND EVERYTHING



ON THE PCW 8256 / 8512



..... BUT WERE AFRAID TO ASK

CP SOFTWARE

READ THIS FIRST

We recommend you make a back-up copy of the master disc before doing anything else. The master disc has been left unprotected for this purpose only. Remember PLEASE BUY DON'T STEAL.

For a practical demonstration of some of the capabilities of All you ever needed to know ..... but were afraid to ask, a DEMO program is supplied on the master disc. To run this proceed as follows:

1. Switch your PCW off and then on again.
2. Insert your CP/M disc (usually side two of the Amstrad discs supplied with the PCW) and press any key. Wait for CP/M to load as evidenced by the A> prompt.
3. Remove the CP/M disc and insert the master disc into drive A.
4. Type DEMO <enter> and the demonstration program should run.

Note that the DEMO program requires the disc to be left in drive A.

After you have finished with the DEMO program, we suggest you go straight to the demonstration section of the manual, which takes you, step by step, through examples designed to get you started using this package.

Introduction

Welcome to ***All you ever wanted to know about GRAPHICS , THE UNIVERSE AND EVERYTHING on the PCW 8256/8512 but were afraid to ask***, programs and documentation designed to help you create graphics output of the standard found only in professional software.

SCODE.COM is a collection of fifty-two machine code routines, including fast, smooth sprite graphics, directly executable from your own programs.

SCODE.GEN is the fully documented Z80 source code for the above routines.

DEMO.COM, DEMO1.COM, DEMO2.COM, DEMO3.COM and various BASIC programs show you some of the many ways the routines can be used.

ERROR.DOC is an ascii file containing documentation about any errors in the manual or program, and information on how to fix them.

The manual contains hitherto unavailable or restricted information, essential for development of good quality software.

1. All the technical and hardware information needed for experienced assembly language programmers to easily produce high quality graphics and stretch the PCW to the limit.

2. Source code and working examples for those with limited knowledge of machine code who want to know more.

3. Compiled and easily accessed machine code routines for direct use by the BASIC programmer who needs fast, smoothly flowing graphic output.

We strongly recommend that you make a back-up copy of the master disc before using it; it has been left without copy protection for this purpose only. **Please Buy, don't Steal.**

DEMONSTRATION

All you ever wanted to know about the PCW 8256/8512 but were afraid to ask is designed to allow you to work up to the standards of the professional programmer. **If you are new to programming** we suggest you start by following this demonstration on your computer, matching it to the instructions below. Just follow the directions and do not be concerned if some of the technical details are obscure, as they are explained later in the manual.
Make sure you type all the instructions accurately, as errors may cause the computer to crash. In this event, don't worry, you can't damage your computer with software crashes, just switch off and start again.

1. At this stage you will need to use the routines under the control of BASIC, so:

a. Turn your PCW off then on again, insert your CP/M disc (usually side two) and press a key. CP/M should load as evidenced by the A> prompt.

b. Insert the master disc into drive A. and type **SCODE <enter>**. The routines will load and announce themselves.

c. Remove the master disc and insert the Amstrad distribution BASIC disc (usually side two). Type **BASIC <enter>** and BASIC should load and announce itself.

d. Finally, type **MEMORY &HBFFF <enter>** to reserve sufficient memory.

2. We will start with a simple example **BEEP**. Beep is routine number (3) and is located at address 49760.
To use it type **A=49760 <enter>**
CALL A <enter>
and you should hear a beep.

You could also use the 'hexadecimal' address C260.
Type **A=&HC260 <enter>**
CALL A <enter>
which should give the same result.

3. You can **turn the beeper on forever** (1) with
A=53902 <enter>
CALL A <enter>
but will probably want to **turn it off** (2) again with
A=53907 <enter>
CALL A <enter>

Incidentally, this technique of turning the beeper on and off, but much faster, is used in routine (48) to emit 'notes' from the speaker.

4. BASIC lacks a **screen clear** command (14) so:
CLS=49749 - note <enter> is assumed from now on.
CALL CLS
should clear the screen for you, except for the **status line,** but you can turn this off (22) with
A=49768
CALL A

5. To **plot a pixel** (19) try:
A=49190
CALL A
You will see a pixel appear at the bottom left corner of the screen, to position the pixel elsewhere on the screen, you'll have to pass some values to the routine.
Lets try an X=50 and Y=100
X=50
Y=100
POKE 49220,X
POKE 49221,0
POKE 49222,Y
POKE 49223,0
CALL CLS - remember this ?
CALL A
and you should see the pixel appear.
To make it disappear (20) again try:
A=58972
CALL A

7. **To draw a line** (17) from X1,Y1 to X2,Y2 lets try:

```
X1=50
Y1=100
X2=100
Y2=200
POKE 56186,1                    ; one line only
POKE 56187,0
POKE 56188,0                    ; origin at 0,0
POKE 56189,0
POKE 56190,0
POKE 56191,0
POKE 56192,X1                   ; use 16 bit value
POKE 56193,0
POKE 56194,Y1
POKE 56195,0
POKE 56196,X2
POKE 56197,0
POKE 56198,Y2
POKE 56199,0
A=49152
CALL A
```

and the line should appear.

8. Interestingly, **to turn the screen off**, (31) type:

```
A=53882
CALL A
```

and you'll see nothing, even though its still there.
Now be careful since you can't see your key presses anymore.
To turn the screen back on (32) type:

```
A=53887
CALL A
```

and all should be clear. If you got in a mess perhaps you'd better switch off and start again !
This routine would be useful for writing to the screen and then showing the full picture.

9. You could **erase the line** (18) drawn previously with:

A=49171

CALL A

since the line coordinate data has not been lost.

10. Now we'll **print various sizes of text** (36), you might like to ensure your printer is on line at this point, since we'll be sending some special graphics there too.

We'll use the text string *HELLO*.

Firstly we need to tell the routine the address at which the text string will be saved and then save the text string there address 58485 (hexadecimal E475) onwards is convenient. E475 is a sixteen bit address and will need to be saved in a special Z80 convention known as low-high. See **Conventions Used** at the end of the manual for an explanation of this.

```
POKE 58352,&H75          ; 16 bit low
POKE 58353,&HE4          ; 16 bit high
POKE 58485,0             ; normal text
POKE 58486,72            ; H
POKE 58487,69            ; E
POKE 58488,76            ; L
POKE 58489,76            ; L
POKE 58490,79            ; O
POKE 58491,36            ; terminator
A=49666
CALL A
```

and HELLO should appear on the screen.

For different text sizes try:

POKE 58485,1

CALL A

and

POKE 58485,2

CALL A

and you should see double height and double width text.

11. You can also **send these characters to the printer** (49). Normally the printer uses its own character set, but this routine uses the same character set as displayed on the PCW screen and therefore allows you to send characters you may have designed youself (38), and to send altered sizes of text.
We'll use the same text as the example above. Make sure your printer is connected !
Note: advance the paper to see the result after each CALL.

```
A=58476
CALL A
POKE 58485,1
CALL A
POKE 58485,0
CALL A
```

and the message should appear in different text sizes.

12. Probably the most versatile routines are those to **plot and move SPRITES** (47). There is data for two simple sprites already in the program, but you will need to *initialise* it (46) first.

```
POKE 53930,1
A=53640
CALL A
```

Then **plot a sprite** on the screen (47) at X,Y (origin at top left)

```
POKE 53996,0
X=200
Y=100
POKE 53943,X
POKE 53944,0
POKE 53945,Y
A=52940
CALL A
```

and the sprite should appear.

To move the sprite to 210,110 (simultaneously deleting the old one)

```
X=210
Y=110
POKE 53943,X
POKE 53944,0
POKE 53945,Y
CALL A
```

and the sprite should move diagonally left.

We leave it to you to examine the BASIC example programs supplied on the disc and try them out. In several cases we have saved 16 bit numbers, if you are unsure of this, please read 'Conventions Used' at the end of this manual, and then retry some of the examples.

Good luck and happy programming.

Loading Instructions

Note that you will always need to have loaded CP/M as described on the first page in order to use the programs on your master disc.

To use the routines directly from BASIC proceed as follows:

1. After the CP/M prompt A>, insert your master disc and type **SCODE <enter>**.
2. The routines will load and announce themselves.
3. Insert your CP/M master disc and type after the CP/M prompt A>, **BASIC <enter>.**
4. BASIC will load and announce itself.
5. Now type **MEMORY &HBFFF <enter>** to protect the routines from corruption by the BASIC program. The routines are ready to be used as desribed in the manual.

Also on the disc are several example BASIC programs. We recommend you **LOAD** these, examine them and try them out. You may wish to use the BASIC routines from line 9000 onwards in your own program by *merging* them into it. The BASIC programs also show you how to send 16 bit parameter values to the routines.

For more experienced programmers, we have also provided the full Z80 source code of the routines. These can be examined using either **Type SCODE.GEN** from CP/M, or examined and modified using an assembler/editor such as the HISOFT DEVPAC80.

Credits

Written by **Chris Whittington** & **Justin Garvanovic, © 1986**
With many thanks to **Electric Studio** & **HiSoft.**

THE COMPILED MACHINE CODE ROUTINES

Supplied on the disc is a file titled SCODE.COM. This contains a whole host of routines that will enable you to do various tasks from both machine code and BASIC.

The majority of the routines assume that information passed to them is *sensible* so care will have to be taken in the form of the data passed. The form of the data required is given in the list of routines. Remember, *Garbage In, Garbage Out.*

Also supplied on the disc is the source code to the routines under the name SCODE.GEN. This was prepared using **HiSoft's Devpac80,** a combined Editor and Assembler. The source code is available for your use to experiment and create more versatile routines.

Now follows a list of all the routines entry points and details of the information needed, and given, by the routines. The routines are all situated in *Common RAM* (C000-FFFF) so that they can be called from and use any RAM currently paged in.

THE ROUTINES

ROUTINE NAME	Start Address in Hex and Decimal	
(1) BEEPER ON	#D28E	(53902)
(2) BEEPER OFF	#D293	(53907)
(3) BEEP	#C260	(49760)
(4) HOME THE CURSOR	#C24A	(49738)
(5) TURN THE CURSOR OFF	#C291	(49809)
(6) TURN THE CURSOR ON	#C296	(49814)
(7) SAVE THE CURRENT CURSOR POSITION	#C29C	(49820)
(8) RESTORE THE CURSOR POSITION	#C2A7	(49831)
(9) MOVE THE CURSOR UP	#C2B2	(49842)
(10) MOVE THE CURSOR DOWN	#C2BA	(49850)
(11) MOVE THE CURSOR LEFT	#C2C2	(49858)
(12) MOVE THE CURSOR RIGHT	#C2CA	(49866)
(13) SET THE CURSOR POSITION	#C2D5	(49877)
(14) CLEAR THE SCREEN	#C255	(49749)
(15) BACK SPACE WITH DELETE	#C27E	(49790)

(16)	DO A CARRIAGE RETURN	#C2F2	(49906)
(17)	DRAW A LINE / SET OF LINES	#C000	(49152)
(18)	DELETE A LINE / SET OF LINES	#C013	(49171)
(19)	PLOT A PIXEL	#C026	(49190)
(20)	UNPLOT A PIXEL	#E65C	(58972)
(21)	FIND THE SYSTEM CLOCK	#C1EC	(49644)
(22)	DISABLE THE STATUS LINE	#C268	(49768)
(23)	ENABLE THE STATUS LINE	#C273	(49779)
(24)	INITIALISE THE RANDOM NUMBER GENERATOR	#C40F	(50191)
(25)	GET A RANDOM NUMBER	#C440	(50240)
(26)	DELAY FOR n SECONDS	#C47E	(50302)
(27)	WAIT FOR FRAME FLYBACK	#D15F	(53599)
(28)	READ THE CASCADE 'JOYCESTICK'	#D242	(53826)
(29)	INVERT THE WHOLE SCREEN	#D26E	(53870)
(30)	RESTORE THE WHOLE SCREEN	#D274	(53876)
(31)	SCREEN OFF	#D27A	(53882)
(32)	SCREEN ON	#D27F	(53887)
(33)	DISC MOTOR ON	#D284	(53892)
(34)	DISC MOTOR OFF	#D289	(53897)
(35)	SYSTEM RESET	#D298	(53912)
(36)	PRINT A STRING	#C202	(49666)
(37)	PRINT n SPACES	#C402	(50178)
(38)	GENERATE A USER DEFINED CHARACTER	#C499	(50329)
(39)	TOGGLE BETWEEN ITALIC & NORMAL CHARACTERS	#C4CB	(50379)
(40)	TOGGLE BETWEEN 2001 & NORMAL CHARACTERS	#C4DE	(50398)
(41)	FILL A SCREEN LINE	#CE24	(52772)
(42)	FLOOD FILL	#E3FE	(58366)
(43)	LOAD A FILE INTO MEMORY FROM DISC	#CD03	(52483)
(44)	SAVE AN AREA OF MEMORY TO DISC	#CD19	(52505)
(45)	GET A CHARACTER	#C20F	(49679)
(46)	INITIALISE THE SPRITE ROUTINE	#D188	(53640)
(47)	PRINT A SPRITE	#CECC	(52940)
(48)	EMIT A 'NOTE' FROM THE SPEAKER	#E444	(58436)
(49)	SEND USER DEFINED GRAPHICS TO THE PRINTER	#E46C	(58476)
(50)	SCROLL THE WHOLE SCREEN	#E681	(59009)
(51)	LOAD A SCREEN FROM DISC	#E6AD	(59053)
(52)	SAVE A SCREEN TO DISC	#E688	(59016)

(1) BEEPER ON (noisy this one) **#D28E (53902)**

Turns the beeper on indefinitely. Needs to be turned off with `BEEPER OFF` (surprise surprise).
Entry conds. - None.
Exit conds. - Corrupts A

(2) BEEPER OFF (thats better) **#D293 (53907)**

Turns the beeper off (dead useful this)
Entry conds. - None.
Exit conds. - Corrupts A

(3) BEEP **#C260 (49760)**

Beeps the speaker for a short time
Entry conds. - None.
Exit conds. - None.

(4) HOME THE CURSOR (poor little thing) **#C24A (49738)**

Sends the cursor to the top left hand corner of the screen.
Entry conds. - None.
Exit conds. - None.

(5) TURN THE CURSOR OFF **#C291 (49809)**

Kills the cursor (aah!).
Entry conds. - None.
Exit conds. - None.

(6) TURN THE CURSOR ON **#C286 (49798)**

Rejuvenates the cursor.
Entry conds. - None.
Exit conds. - None.

(7) SAVE THE CURRENT CURSOR POSITION **#C29C (49820)**

Saves the current screen position of the cursor.
Entry conds. - None.
Exit conds. - None.

(8) RESTORE THE CURSOR POSITION **#C2A7 (49831)**
Gets the saved cursor position back.
Entry conds. - None.
Exit conds. - None.

(9) MOVE THE CURSOR UP ONE POSITION **#C2B2 (49842)**
Move the cursor up a square
Entry conds. - None.
Exit conds. - None.

(10) MOVE THE CURSOR DOWN ONE POSITION **#C2BA (49850)**
Move the cursor down a square
Entry conds. - None.
Exit conds. - None.

(11) MOVE THE CURSOR LEFT ONE POSITION **#C2C2 (49858)**
Move the cursor left a square
Entry conds. - None.
Exit conds. - None.

(12) MOVE THE CURSOR RIGHT ONE POSITION **#C2CA (49866)**
Move the cursor right one square
Entry conds. - None.
Exit conds. - None.

(13) SET THE CURSOR POSITION **#C2D5 (49877)**
Place the cursor absolutely anywhere you want on the screen. (honestly)
Entry conds. - #C2F0 (49904) -> Row number.
#C2F1 (49905) -> Column number.
Exit conds. - None.

(14) CLEAR THE WHOLE SCREEN **#C255 (49749)**
You are not going to believe what this does.
Entry conds. - None.
Exit conds. - None.

(15) BACK SPACE WITH DELETE **#C27E (49790)**

Delete the character to the left of the cursor and move the cursor left one square.

Entry conds. - none.

Exits conds. - none.

(16) DO A CARRIAGE RETURN **#C2F2 (49906)**

Send a carriage return to the screen.

Entry conds. - None.

Exit conds. - None.

(17) DRAW A LINE OR SET OF LINES **#C000 (49152)**

Will draw a line, or a number of lines, between any positions on the screen.

Entry conds. - #DB7A,#DB7B (56186,56187) = amount of lines to be drawn. In the range 1 - 270.

#DB7C,#DB7D	(56188,56189)	= X origin.
#DB7E,#DB7F	(56190,56191)	= Y origin.
#DB80,#DB81	(56192,56193)	= First X position
#DB82,#DB83	(56194,56195)	= First Y position
#DB84,#DB85	(56196,56197)	= Second X position
#DB86,#DB87	(56198,56199)	= Second Y position
	(data for the next line/s.)	
#DB88,#DB89	(56200,56201)	= First X position
#DB8A,#DB8B	(56202,56203)	= etc. etc. etc.

Position (0,0) is at the bottom left of the screen with (719,255) at the top right hand corner.

If for instance you wished to draw a line between these two points the data sent would look like this:-

Location	Contents	Comments
#DB7A,#DB7B	1,0	Set the number of lines to one.
#DB7C-#DB7F	0	Set the origin to (0,0)
#DB80-#DB83	0	Set start point to (0,0)
#DB84,#DB85	207,2	Set X position to 719 (207+2*256)
#DB85,#DB86	255,0	Set Y position to 255 (255+0*256)

Exit conds. - AF,HL,DE,BC,C' are all corrupted.

(18) DELETE A LINE OR A SET OF LINES **#C013 (49171)**

This will erase a line, or set of lines, between any screen positions.
Entry conds. - (Exactly the same as for DRAW LINE)
Exit conds. - AF,HL,DE,BC,C' are all corrupted.

(19) PLOT A PIXEL **#C026 (49190)**

Plots a pixel anywhere on the screen.
Entry Conds. - #C044,#C045 (49220,49221) -> X pixel position.
#C046,#C047 (49222,49223) -> Y pixel position.
Exit conds. - AF,DE,HL,BC are all corrupted.

(20) UNPLOT A PIXEL **#E65C (58972)**

Erase a pixel form anywhere on the screen.
Entry Conds. - (as for plot pixel)
Exit conds. - AF,DE,HL,BC are all corrupted.

(21) FIND THE SYSTEM CLOCK **#C1EC (49644)**

Send back the address of the system clock.
Entry conds. - None.
Exit conds. - #C200,C201 (49664,49665) point to the address.
AF,DE,BC are all corrupted, HL holds the address.

(22) DISABLE THE STATUS LINE **#C268 (49768)**

Get rid of the line at the bottom of the screen.
Entry conds. - None.
Exit conds. - None.

(23) ENABLE THE STATUS LINE **#C273 (49779)**

Bring the status line back to life.
Entry conds. - None.
Exit conds. - None.

(24) INITIALISE RANDOM NUMBER GENERATOR **#C40F (50191)**

Sets the initial seed for the random number generator. This will have to be called before the random number generator is used.
Entry conds. - None.
Exit conds. - None.

(25) GET A RANDOM NUMBER **#C440 (50240)**

Gets a random number between 0 & 255.
Entry conds. - None.
Exit conds. - #C477 (50295) holds the random number.
A holds the random number.

(26) DELAY FOR n SECONDS **#C47E (50302)**

Wait around and do nothing for n seconds.
Entry conds. - #C498 (50328) holds the delay in seconds.
Exit conds. - None.

(27) WAIT FOR A FRAME FLYBACK **#D15F (53599)**

Wait for the electron gun to start redrawing the screen.
Entry conds. - None.
Exit conds. - AF corrupted.

(28) READ THE CASCADE 'JOYCESTICK' **#D242 (53826)**

Read the joystick and set one of 5 memory locations depending on which position the stick is in.
Entry conds. - None.
Exit conds. - #D29D (53917) -> Joystick left (1=button pressed)
#D29E (53918) -> Joystick Right (1=button pressed)
#D29F (53919) -> Joystick up (1=button pressed)
#D2A0 (53920) -> Joystick down (1=button pressed)
#D2A1 (53921) -> joystick fire (1=button pressed)
AF,B are corrupted.

(29) INVERT THE WHOLE SCREEN **#D26E (53870)**

Inverts the whole screen, including the border.
IMPORTANT. For this routine to work it must disable the interrupts so great care should be taken with it.
Entry conds. - None.
Exit conds. - A gets corrupted.

(30) RESTORE THE WHOLE SCREEN **#D274 (53876)**

Restore the screen to it's normal state after it has been inverted.
Entry conds. - None.
Exit conds. - A gets corruputed.

(31) SCREEN OFF (not literally) **#D27A (53882)**

Turns the whole screen off. Useful for drawing something with the screen off and making it instantly re-appear by turning it back on again.
Entry conds. - None.
Exit conds. - A gets corrupted.

(32) SCREEN ON **#D27F (53887)**

Turns the screen back on after it has been switched off.
Entry conds. - None.
Exit conds. - A gets corrupted.

(33) DISC MOTOR ON **#D284 (53892)**

Turns the disc motor on (bet you didn't expect that).
Entry conds. - None.
Exit conds. - A gets corrupted.

(34) DISC MOTOR OFF **#D289 (53897)**

Opens the garage door ! (after turning the disc motor off).
Entry conds. - None.
Exit conds. - A gets corrupted.

(35) SYSTEM RESET **#D298 (53912)**

Does a complete reset
Entry conds. - None.
Exit conds. - What exit ?

(36) PRINT A STRING **#C202 (49666)**

Will print a string at the current cursor position in any one of 3 text sizes.
Entry conds. - #E3F0,#E3F1 (58352,58353) -> holds the address
that points to the start of the message.
The message should be terminated with a '$' sign.
If certain values are inserted into the string then
the size of the text can be altered.
The values are 0-> Normal size text.
1-> Double width text.
2-> Double width and height text.
Exit conds. - AF,BC,DE,HL all get corrupted.

(37) PRINT n AMOUNT OF SPACES **#C402 (50178)**

This will print a certain number of spaces at the current cursor position.
Note that if the 'PRINT STRING' function was last used
to print large text then the spaces will be the size of the text.
Entry conds. - #C40E (50190) -> The amount to be printed.
Exit conds. - AF,BC,DE,HL are all corrupted.

(38) GENERATE A USER DEFINED CHARACTER **#C499 (50329)**

Re-define any of the characters to something of your choice.
Entry conds. - #C4C2 (50370) should contain 9 bytes of data.
The first value should contain the character that is
going to be defined. The next 8 bytes should
contain the data.
Exit Conds. - AF,DE,BC,HL are all corrupted.

(39) TOGGLE ITALIC / NORMAL CHARACTER SET **#C4CB (50379)**

Toggle between the italic and normal character set.
Entry conds. - None.
Exit conds. - AF,BC,DE,HL are all corrupted.

(40) TOGGLE 2001 / NORMAL CHARACTER SET **#C4DE (50398)**

Toggle between the 2001 and normal character set.
Entry conds. - None.
Exit conds. - AF,BC,DE,HL are all corrupted.

(41) FILL A SCREEN LINE **#CE24 (52772)**

Draw a line outwards from a specified point until an obstruction is found
or the edge of the screen is met.
Entry conds. - #DB73 (56179) -> Y coordinate (from top left hand corner.)
#DB74,#DB75 (56180,56181) -> X coordinate.
Exit conds. - AF,BC,DE,HL are all corrupted.

(42) FLOOD FILL **#E3FE (58366)**

Fill an area of screen from a set point outwards. The fill will work
outwards until an obstruction is found and then move up until an
obstruction is found and finally work down until an obstruction is found.
Entry conds. - (as for the line fill)
Exit conds. - AF,BC,DE,HL are all corrupted.

(43) LOAD A FILE INTO MEMORY **#CD03 (52483)**

Will load a named file from any drive to anywhere in memory. It's best use is in pulling files from the ram disc into memory quickly.

Entry conds. - #CCFF,#CD00(52479,52480) -> Should hold the first address the data will be going in to.

#CD01,#CD02 (52481,52482) -> Should hold the last address the data will go into.

#CE18 (52760) onwards -> Should hold the filename in the following format:

First character -> Drive (A,B,M).

The next 8 characters hold the filename padded out with spaces. We recommend you use upper case A-Z only.

The next 3 characters hold the file type. eg. COM.

Exit conds. - AF,BC,DE,HL are all corrupted.

(44) SAVE AN AREA OF MEMORY TO DISC **#CD19 (52505)**

Will save any area of memory to disc.

Entry conds. - (As used in LOAD FILE)

Exit conds. - AF,BC,DE,HL are all corrupted.

(45) GET A CHARACTER **#C20F (49679)**

Read a character from the keyboard. Accepts keys 0-9, A-Z and RETURN. Lower case letters are converted to upper case.

Entry conds. - None.

Exits conds. - #C249 (49737) contains the key value.

A=key value.

(46) SPRITE INITIALISE **#D188 (53640)**

Initialises the sprite data for the sprite print routines.

Entry conds. - #D9ED (55789) -> The sprite data should be put in here in the following form.
byte 1 -> width of sprite in bytes.
byte 2 -> height of sprite in pixels.
byte 3 onwards should contain the data.
There should only be (width)*(height) amount of data.
(If there are 2 sprites it's data should come directly after the first one's.)
N.B. The maximum sprite size is 6 bytes wide by 32 pixels high.
There is no detection within the program to detect whether this has been exceeded.

#D2AA (53930) -> contains the number of sprites that need initialising, should be either 1 or 2.

Exit conds. - AF,BC,DE,HL are all corrupted.

(47) PRINT A SPRITE **#CECC (52940)**

Prints a sprite on the screen and deletes an old image if there is one.
This routine makes use of the frame flyback routine to allow smooth movement.

Entry conds. -

#D2EC (53996) -> Number of the sprite to be moved. Should be either 0 or 1, any value greater than 1 is treated as a 1.

#D2B7,#D2B8 (53943,53944) -> Holds the X position of sprite 0 (0 - 719)

#D2B9 (53945) -> Holds the Y position of Sprite 0 (0-255), from the top left hand corner.

#D2D7,#D2D8 (53975,53976) -> Holds the X position of sprite 1 (0 - 719)

#D2D9 (53977) -> Holds the Y position of sprite 1 (0-255), from the top left hand corner.

Exit conds. - AF,BC,DE,HL are all corrupted

PLEASE NOTE: TO MAKE LIFE EASIER WHERE THE SPRITE ROUTINES ARE CONCERNED THERE ARE ALREADY A SET OF 2 SPRITES IN THE SOURCE. YOU CAN EXAMINE THEM TO SEE THEIR LAYOUT AND GENERALLY JUST PLAY WITH THEM.

(48) PRODUCE A 'NOTE' FROM THE SPEAKER #E444 (58436)

This routine tries to produce a 'note!' out of the bleeper

Entry conds. - #E46A (58474 -> This should contain the pitch
#E468 (58472) -> This should contain the duration

Exit conds. - AF,HL,DE,BC are all corrupted

(49) SEND USER DEFINED GRAPHICS TO PRINTER #E46C (58476)

This routine will allow you to send user defined graphics to the printer.
It will also allow you to send double height andwidth characters to the printer as well.

Entry conds. - A string of not more than 80 characters should be sent stored at the address #E475 (58485). The first character in the message should be a 'control code' to determine which size of text is to be used. These codes are : 0 -> Normal size
1 -> Double width
2 -> Double width and height
The message should be terminated with an ascii $ sign (36)

Exit conds. - AF,HL,DE,BC are all corrupted

(50) SCROLL THE WHOLE SCREEN UP OR DOWN # E681 (59009)

Scroll the whole screen up or down a user specified amount

Entry Conds. -#E687 (59015) contains the displacement that the screen is to be moved.

Exit conds. - A is corrupted

(51) **LOAD A SCREEN FROM DISC # E6AD (59053)**

Load a previously saved screen from disc.

Entry Conds. The filename is stored at #E6D1 (59089) onwards.

Exit Conds. All registers corrupted.

(52) **SAVE A SCREEN TO DISC # E688 (59016)**

Save the screen to disc.

Entry Conds. The filename is stored as above.

Exit Conds. All registers corrupted.

ROLLER RAM & SCREEN RAM

Roller ram is an area of memory that holds 256 pointers to the start of each video screen line. That mild description hardly does justice to the power of this unique feature. For instance by simply moving the pointers to an adjacent address the whole screen can be made to scroll up and down, pixel by pixel, at will. This can be applied to the whole screen or just parts of it.

When the screen is hard scrolled by moving the cursor off the bottom of the screen the roller ram is also updated which means that any access to the screen must be done *via* the roller ram, rather than assuming a well organised screen and calculating the screen address. Every routine that's supplied with this package goes through roller ram to get a screen address, this even includes the sprite routines. If greater speed is needed it would be possible to reset the screen and access it directly assuming that no scrolling had been done.

Other effects can also be produced by manipulating the roller ram such as echoing an area of the screen to another by simply copying one area of roller ram to another. Multi-directional scrolling is possible, one area going up and another going down, as seen at the end of the demo.

It is possible to point roller ram to other areas of memory. This could be useful for updating say line drawn graphics quickly by drawing them on a hidden screen and then pointing roller ram to the hidden screen.

The actual screen layout is fairly straightforward. The diagram below shows how the top left hand corner of the screen is arranged.

	COLUMN 1	**COLUMN 2**	**COLUMN 3**	
ROW 1	START+0	START+8	START+16	etc.
ROW 2	START+1	START+9	START+17	etc.
ROW 3	START+2	START+10	START+18	etc.
ROW 4	START+3	START+11	START+19	etc.
ROW 5	START+4	START+12	START+20	etc.
ROW 6	START+5	START+13	START+21	etc.
ROW 7	START+6	START+14	START+22	etc.
ROW 8	START+7	START+15	START+23	etc.
ROW 9	START+720	START+728	START+736	
	etc	etc	etc	

(this assumes that no scrolling has been done).

COMMON RAM

Common ram is the name given to the area of memory from #C000 - #FFFF. It is called common ram because whatever area of lower ram is paged in this area always remains paged in. This makes it the ideal place to put routines that will directly access the screen or roller ram. Not all of the memory from #C000 - #FFFF is available to the user as CP/M takes the memory from about #F000 upwards so care should be taken not to overwrite any of the important code above #F000.

SCROLLING

As already mentioned it is possible to scroll the whole screen or parts of it up and down using roller ram. This can be pixel by pixel, or more if needed. It is also possible to scroll the screen sideways a byte at a time using roller ram. This would be accomplished by taking an address from the roller ram, incrementing or decrementing it, and putting it back. There is however a slight problem with roller ram that has not yet been discussed. The addresses stored in roller ram do not point directly to the screen address, but first require some decoding. The routine that does this can be found in the source listing - SCODE.GEN. The scrolling routine supplied in the source will only operate on the whole screen. Please refer to the short listing below that uses roller ram to scroll the screen, with modification it would be possible to scroll not just the whole screen but parts of it by this method.

```
        LD    IX,#B600      ; The roller ram address of the
                            ; top line of the screen.
        LD    DE,#B602      ; The roller ram address of the
                            ; second screen line down.
        LD    L,(IX+0)      ; Get the low byte of the value
                            ; in the top line of roller ram.
        LD    H,(IX+1)      ; Get the high byte of the value
                            ; in the top line of roller ram.
        LD    B,255         ; This is the amount of lines
                            ; we want to scroll.
SCR10   LD    A,(DE)        ; Get the low byte from the
                            ; second roller ram line down.
        LD    (IX+0),A      ; Put it in the low byte of the
                            ; top lines roller ram.
        INC   DE            ; Move the pointer on to the
                            ; high byte of the roller ram.
        INC   IX            ; Move the pointer on to the
                            ; high byte of the roller ram.
        LD    A,(DE)        ; Get the high byte from the
                            ; second line of roller ram.
        LD    (IX+0),A      ; Put it into the high byte of
                            ; the top lines roller ram.
        INC   DE            ; Move the pointer on to the low
                            ; byte of the next address.
        INC   IX            ; Move the pointer on to the low
                            ; byte of the next address.
        DJNZ  SCR10         ; Have we done all the lines yet
                            ; If not then jump back
        LD    (IX+0),L      ; Put the low byte of the
                            ; original top line address back.
        LD    (IX+1),H      ; Put the high byte of the
                            ; original top line address back.
        RET                 ; And that finishes it.
```

There is though another method of scrolling the whole screen that does not use roller ram and is virtually instant. This involves the use of one of the output ports, more information on these can be found elsewhere. Port #F6 holds a screen offset value that when adjusted forces the whole screen to move up or down depending on whether the value was incremented or decremented. By adding say 128 to the value of the port it causes the screen to be shifted so that the top line now appears in the middle. The only draw back with this method though is that the whole screen is scrolled as opposed to any amount by the roller ram method.

ACCESSING SCREEN RAM

If you wish to access the screen ram, roller ram or character ram directly a special routine is needed to page in the screen environment. This will jump to your program, which must be in common ram (#C000-#FFFF), and when it finishes it will return back to where ever the routine is. The paging routine is listed below.

```
        LD      HL,(1)          ; Get address of user function
        LD      BC,87           ; Get displacement
        ADD     HL,BC           ; -> HL to the user function
                                ; entry point.
        LD      (ENTRY+1),HL    ; Store this address so that
                                ; it forms a call (HL).
        LD      BC,PROG         ; -> BC to the start of your
                                ; program in common ram.
ENTRY   CALL    0000            ; This will now have become
                                ; a CALL (HL) statement.
        DEFW    #00E9           ; This is the value of the
                                ; routine that puts you into
                                ; the screen enviroment.
                                ; Note that it is a 16 bit
                                ; value.
        RET                     ; Once your program has ended
                                ; and RETurned it will end up
                                ; at this instruction. This
                                ; could be anywhere in memory.
```

Input & Output Ports

The ports on the PCW can be made to perform some powerful tasks from turning the screen on and off to scrolling the whole screen. Of all that's known, or not known as the case may be, the ports are probably the most mysterious. The information printed here has taken a lot of people a long time to accumulate and still leaves plenty of room for more exploration.

PORT	FUNCTION/S
#F0	This port is used to read the Cascade joystick if it is attached.
#F5	This port holds the screenbase of the roller ram divided by 2. Useful for creating alternate video RAM pages.
#F6	This port holds the horizontal screen offset and altering it will cause the whole screen to scroll up or down.
#F8	This would appear to be a general purpose port as it has many uses both for input and output. It's input function is the detection of line or frame flybacks. It is useful to wait for a frame flyback before scrolling or moving sprites as it will make the action appear smooth. I can't really think of a use for the line flyback detection as its happening so fast but its there anyway. To detect these events read bit 6 and if a 1 is found a line flyback has occured. If you then read the port straight after detecting the line flyback and find bit 6 still containing 1 then a frame flyback has occured.

The output side of this port has many varied uses

VALUE	EFFECT
01	Cold Boot (be careful !).
07	Turn the screen on.
08	Turn the screen off.
09	Turn the disc motor on.
10	Turn the disc motor off.
11	Turn the bleeper on.
12	Turn the bleeper off.

EDGE CONNECTOR

These are the pin-outs for the edge connector. All will be recognised by those with Z80 hardware knowledge except perhaps **MDIS**, which disables all internal devices attached to the Z80.

T O P

No connection	(1) [ ] (2)	No connection
0V	(3) [ ] (4)	0V
+5V	(5) [ ] (6)	+5V
No connection	(7) [ ] (8)	+12V
A14	(9) [ ] (10)	A15
A12	(11) [ ] (12)	A13
A10	(13) [ ] (14)	A11
A8	(15) [ ] (16)	A9
A6	(17) [ ] (18)	A7
A4	(19) [ ] (20)	A5
A2	(21) [ ] (22)	A3
A0	(23) [ ] (24)	A1
D6	(25) [ ] (26)	D7
D4	(27) [ ] (28)	D5
D2	(29) [ ] (30)	D3
D0	(31) [ ] (32)	D1
RESET	(33) [ ] (34)	MI
BUSRQ	(35) [ ] (36)	INT
BUSAK	(37) [ ] (38)	WAIT
WR	(39) [ ] (40)	MREQ
RD	(41) [ ] (42)	IORQ
No connection	(43) [ ] (44)	NSYNC
MDIS	(45) [ ] (46)	VIDEO
3.2 MHZ	(47) [ ] (48)	Z80CLK (4MHZ)
0V	(49) [ ] (50)	0V

Further Reading

For information on the CP/M operating system we recommend the book:
OSBORNE CP/M USER GUIDE by Thom Hogan published by Osborne/McGraw Hill.

A book we recommend for beginners in Z80 assembly language is:
PROGRAMMING THE Z80 by Rodnay Zaks published by Sybex.

We hope that you will find this package useful and in order to encourage software development on the PCW, we offer to evaluate and consider publication of high quality software containing routines from this package.

Technical support

If you have a problem, or think you may have found a bug, then it is always easier for us to deal with your query if you write to us at:

Technical Support
CP Software, Stonefield, The Hill, Burford
Oxfordshire OX8 4HX

Conventions Used

The source code was prepared using **HiSoft's Devpac80**. The normal modifications will probably have to be done to the source to make it compatable with any other assemblers. In general the symbol # (hash) before a number implies a hexadecimal base.

Some of the data required by the routines is in the form of a 16 bit value. You will need to know how to convert a 16 bit number into two 8 bit numbers, here is one method. This will take a 16 bit number X and produce the value L (low byte) and H (high byte). Lets assume that X is 54321, first divide X by 256. This gives the answer 212.1914062, which should be rounded down to 212, which is the value H. Now times 212 by 256 and take the answer from 54321. This gives the answer 49, which is the value for L. So 212*256+49 = 54321

Examples. 400 = 1 (high byte) , 144 (low byte) = 1*256+144=400
49152 = 192 (high byte) , 0 (low byte) = 192*256+0=49152
(Please note that all 16 bit values being passed into the routines should be stored with the low byte first followed by the high byte, this being the normal Z80 convention, and contrary to common sense!)