P. DAVIDSON

WILDCARDS

Volume Three

A collection of programs, tips and techniques for all models of the

MICROBEE®

personal computer

BURT : FORD : NALLAWALLA

WILDCARDS
VOLUME THREE

TIPS, TECHNIQUES AND PROGRAMS

FOR THE

MICROBEE PERSONAL COMPUTER

by

R.A. Burt

P.T. Ford

A. Nallawalla

and others

Published and distributed by:
BF&N Publishing, P.O. Box 85, Williamstown, VIC 3Ø16, Australia.

FIRST EDITION

NOTE:

In the text, wherever reference is made to hardware or software, the comments are aimed at the following models/versions:

ITEM		MODEL/VERSION
EDASM	:	ROM, 4.2 and 4.3
MICROBEE	:	8k, 16k, 32k, 64k and 128k*
MICROBEE MONITOR	:	1.Ø, 1.1, Net and Disk
MICROWORLD BASIC	:	5.1Ø, 5.22e and 6.22e
WORD-BEE	:	1.Ø, 1.2 and Disk
WORDSTAR	:	3.Ø and 3.3

***NOTE:** Newer models have names such as Educator, Experimenter etc. This book is generally applicable to all models, with all exceptions mentioned where known.

ISSN: Ø811-577X Paperback

Printed in Australia by List Print, Geelong, Victoria.

C O N T E N T S

CREDITS

CP/M is a Registered Trademark of Digital Research Inc.
MICROBEE is a Registered Trademark of Microworld (1982).
MICROWORLD BASIC is the Copyright of Matthew Starr.
MICROSOFT is a Registered Trademark of Microsoft Inc.
WORD-BEE is the Copyright of R.C. Harris.
WORDSTAR and MAILMERGE are the Registered Trademarks of MicroPro
Z8Ø is a Registered Trademark of Zilog Inc.

INTRODUCTION

Over 20,000 units of the MicroBee personal computer have been sold since March 1982. Manufactured by Applied Technology Pty Ltd, of Sydney NSW, it was sold initially to hobbyists only in kit form for $A399.00. Today it is sold fully-assembled and tested in several configurations: 8k/16k/32k/64k/128k of Random Access Memory, colour or monochrome, disk or cassette storage. It is approved by Education Departments in Queensland, New South Wales and Western Australia. MicroBees have been exported to many overseas countries.

This book was partly compiled from contributions from other users. It was written with the aid of WordStar, the leading word processing program that is supplied with the Applied Technology disk system, whilst program listings have been merged with the text through software control to avoid risk of typographical errors.

The WILDCARDS series have been selling well not only in Australia and New Zealand, but in several overseas countries as well, particularly Scandinavia. This parallels the relative success of the MicroBee in those countries.

WILDCARDS Volume Three is a book, not a magazine, so if this is your first look at the series, you may wish to buy the previous volumes to complete the picture. Each volume covers several topics, but the underlying theme is always 'Tips and Techniques'. Each book contains programs and information that will permit the MicroBee owner to get better value from the computer. It is aimed at the intermediate user who has finished reading the User's Manual, written a few programs and is expecting more from his or her MicroBee. Although the beginner may feel alarmed at seeing many assembler listings in this volume, we have included them to give people a further taste of machine language. A BASIC loader is usually supplied for the benefit of those who have not yet bought the Editor/Assembler or simply wish to use the program without knowing how it works.

APPLICATIONS

INTRODUCTION

This chapter contains programs that illustrate some practical applications of the techniques described in this book or in WILDCARDS Volumes One or Two.

LARGE CHARACTERS USING LORES AND HIRES GRAPHICS

by Robin Clipsham

One advantage of the Microbee's graphics implementation, using a Programmable Character Generator (PCG), is the ability to define alternative character fonts. It is, however, a long and tedious task to redefine each character. One alternative is to use data already stored in the character generator ROM to define characters of varying sizes in either LORES or HIRES graphics. A number of sample programs and subroutines follow which demonstrate some of the possible techniques which you might find useful for your own applications.

In order to access the data contained in the character generator ROM it is necessary to set the ROM Read Latch by setting bit Ø of Port ØB. Unfortunately BASIC resets the Latch before we are able to access the character generator, however we can overcome this problem by using a machine language subroutine.

The following subroutine loads the 16 Bytes of data for a specified character into an integer array Z(15) which must have been dimensioned prior to entry.

Program name : ROMRDS

```
00100 ;This Subroutine returns the contents of the character
00110 ; generator ROM for the ASCII code supplied in C reg
00120 ; as 16 integer values in the array Z(15).
00130 ;
00140          DEFR    16D
00150          ORG     5000H    ;OR ANYWHERE YOU WISH
00160 ZPTR     EQU     06E2H    ;POINTER TO Z
00170          LD      HL,(ZPTR)
00180          LD      A,0FFH
00190          CP      (HL)     ;CHECK ARRAY FLAG
00200          RET     NZ       ;ARRAY NOT HERE
00210          INC     HL
00220          INC     HL
```

```
00230           INC     HL
00240           LD      A,1FH
00250           CP      (HL)    ;CHECK ARRAY SIZE
00260           RET     NC      ;TOO SMALL
00270           INC     HL
00280           EX      DE,HL   ;DEST IN DE
00290           LD      L,0
00300           LD      H,C     ;ASCII CHARACTER
00310           LD      B,4
00320 SHIFT     SCF
00330           RR      H
00340           RR      L       ;CALC ROM ADDRESS
00350           DJNZ    SHIFT
00360           LD      B,10    ;16 BYTES TO LOAD
00370           LD      A,1
00380           OUT     (0BH),A ;SET ROM READ LATCH
00390 LDARRY    LD      A,(HL)
00400           LD      (DE),A
00410           INC     HL
00420           INC     DE
00430           INC     DE
00440           DJNZ    LDARRY
00450           RET
```

An example of the use of this subroutine is a short BASIC program to create large characters in LORES graphics which can be scrolled across the screen in a similar manner to commercial advertising displays. We need to use a further short machine language subroutine to scroll the screen horizontally.

```
00100 ;This Subroutine scrolls screen sideways 1 position
00110           ORG     5100            ;OR WHEREVER YOU LIKE
00120           LD      HL,0F000        ;START OF SCREEN RAM
00130           LD      DE,0040         ;64 COLUMNS
00140           LD      B,10            ;16 LINES
00150 CLR1PS    LD      (HL),20         ;CLEAR FIRST POSITION OF EACH LINE
00160           ADD     HL,DE
00170           DJNZ    CLR1PS
00180           LD      HL,0F001
00190           LD      DE,0F000
00200           LD      BC,03FFH
00210           LDIR                    ;SHIFT SCREEN LEFT ONE POSITION
00220           RET
00230           END
```

BASIC LOADER - LORES VERSION

The Machine Language Subroutines are contained in lines 10 - 14 and 20 - 22 and are poked into memory locations reserved by the REM statements in lines 1 and 2. Lines 3 to 22 can be deleted

after the machine code has been loaded and the program can then be saved to cassette. (EDITORS' NOTE: In line nos 00001 and 00002 do NOT split up the lines as shown or else the embedded routines may not work. We have done so for formatting purposes only. Leave no space after the second '!'.)

Program name : ADVERT

```
00001 REM This line contains ROMREAD subroutine ---------!---------!
      ---------!---------!--
00002 REM This line contains HZSCROL subroutine:---------!---------!
      -----
00003 FOR I=0 TO 41:READ H:POKE 2348+I,H:NEXT I
00004 FOR I=0 TO 24:READ H:POKE 2435+I,H:NEXT I
00010 DATA  42,226,  6, 62,255,190,192, 35, 35, 35
00011 DATA  62, 31,190,208, 35,235, 46,  0, 97,  6
00012 DATA   4, 55,203, 28,203, 29, 16,249,  6, 16
00013 DATA  62,  1,211, 11,126, 18, 35, 19, 19, 16
00014 DATA 249,201
00020 DATA  33,  0,240, 17, 64,  0,  6, 16, 54, 32
00021 DATA  25, 16,251, 33,  1,240, 17,  0,240,  1
00022 DATA 255,  3,237,176,201
00100 DATA 64,128,16,32,4,8,1,2
00110 DIM Z(15):                              REM Array for ROM Data
00120 CLS:LORES
00130 CURS 1,15:INPUT "Enter String to be displayed ";A0$
00140 FOR K=1 TO LEN(A0$)
00150 A=USR(2348,ASC(A0$(;K,K))):             REM Call ROMREAD routine
00160 FOR G = 0 TO 3:READ H,J:                REM Test 2 bits at a time
00170 FOR I = 4 TO 15:T=Z(I):                 REM Scan lines 4 - 15
00180 B=(TANDH):IF B>< 0 THEN SETH 127,I: REM Set point if bit on
00190 B=(TANDJ):IF B>< 0 THEN SETH 126,I: REM      (DITTO)
00200 NEXT I
00210 A=USR(2435):NEXT G:                     REM Call hor. scroll subr.
00220 RESTORE 100:NEXT K
00230 GOTO 140
00240 END
```

BASIC LOADER - HIRES VERSION

The same subroutine is used in the following program to read the character ROM in order to generate double size characters in HIRES graphics. (Do NOT put spaces in line 00001 after the second '!' as shown.)

Program name : HRS2XB

```
00001 REM This line contains ROMREAD subroutine:---------!---------!
      ---------!---------!--
00003 FOR I=0 TO 41:READ H:POKE 2348+I,H:NEXT I
00010 DATA 42,226,6,62,255,190,192,35,35,35
00011 DATA 62,31,190,208,35,235,46,0,97,6
00012 DATA 4,55,203,28,203,29,16,249,6,16
00013 DATA 62,1,211,11,126,18,35,19,19,16
00014 DATA 249,201
00100 DATA 128,64,32,16,8,4,2,1
00110 DIM Z(15):                    REM Array for ROM data
00120 CLS:CURS 1,15:INPUT "Enter String to be displayed ";A1$
00130 CLS:HIRES:L=0:C=0
00140 FOR K=1 TO LEN(A1$)
00150 IF C <31 THEN LET C=C+1 ELSE LET C = 0:L=L+1: REM New line
00160 GOSUB[A1$(;K,K),L,C] 250:     REM Call Display subroutine
00170 NEXT K
00180 END
00190 REM ******************************************************
00200 REM * This subroutine displays the character provided   *
00210 REM * as parameter 1 at line/column specified by        *
00220 REM * parameters 2 and 3 . The screen contains 8 lines  *
00230 REM * of 32 characters each.                            *
00240 REM ******************************************************
00250 VAR(A0$,L,C)
00260 A=USR(2348,ASC(A0$)):         REM Call ROMREAD Subroutine
00270 FOR J = 0 TO 15:              REM 16 data bytes
00280 T=Z(J):IF T=0 THEN 330
00290 FOR I = 0 TO 7:               REM Test each bit in turn
00300 X=16*C+2*I:Y=32*L+2*J:READ H
00310 B= (T AND H): IF B<> 0 THEN SETH X,Y: SETH X+1,Y:SETH X,Y+1:
      SETH X+1,Y+1:                 REM Set 4 HIRES points if bit on
00320 NEXT I
00330 RESTORE 100:NEXT J
00340 RETURN
00350 END
```

You will see that the process of SETting a large number of HIRES points can become fairly slow in BASIC, so the following machine language routine is provided to display a complete, double size character. As with all HIRES graphics, there is always the chance of running out of PCG characters, so the BASIC program should check for this condition in the same manner as the sample BASIC driver. Each different character displayed uses up to a maximum of five PCG characters, allowing a total of at least 25 different characters to be displayed. The position of the characters on the screen may be controlled using the BASIC CURS command.

Program name : 2XASM

```
00100 ;*******************************************************
00110 ;*  This subroutine displays a double size character    *
00120 ;* on the screen at the position specified by the BASIC *
00130 ;* cursor pointer at 10BH . The character to display    *
00140 ;* is provided as a parameter in tne C register and the *
00150 ;* cursor pointer is updated prior to exit.            *
00160 ;*******************************************************
00170         DEFR    16D
00180         ORG     5000
00190 CRSLOC  EQU     10BH
00200         LD      HL,(CRSLOC)     ; Get cursor address
00210         LD      A,3FH
00220         AND     L       ; Get cursor column in A
00230         LD      D,0
00240         LD      E,A
00250         XOR     A       ; Reset carry
00260         RL      E       ; Shift DE left
00270         RL      E       ; four times to
00280         RL      E       ; multiply by 8
00290         RL      D       ; giving HIRES X-coordinate
00300         PUSH    DE
00310         LD      A,0C0H
00320         AND     L       ; Get L.S.Bits of cursor line #
00330         SRL     A
00340         SRL     A
00350         LD      E,A
00360         LD      A,03
00370         AND     H       ; Get M.S.Bits of cursor line #
00380         RRCA
00390         RRCA
00400         OR      E       ; Merge with L.S.Bits
00410         CPL             ; Invert ( Y axis runs up screen )
00420         LD      E,A
00430         LD      D,0     ; DE now has cursor Y coordinate
00440         LD      L,0
00450         LD      H,C     ; ASCII Char passed as parameter
00460         LD      B,4
00470 SHIFT   SCF             ; Convert to ROM address
00480         RR      H       ; by shifting right 4 times
00490         RR      L       ; and adding F000 ( carry F/F )
00500         DJNZ    SHIFT
00510         PUSH    HL
00520         POP     IX      ; Transfer ROM address to IX
00530         POP     HL      ; Pop X coordinate off stack
00540         LD      B,10    ; 16 Bytes of ROM
00550 NXTBYT  PUSH    HL      ; Save X coord.
00560         LD      C,80    ; C contains bit mask
00570         LD      A,1
00580         OUT     (0BH),A ; Set ROM read latch
00590         LD      A,(IX+0)
00600         PUSH    AF      ; Save ROM character
00610         XOR     A
```

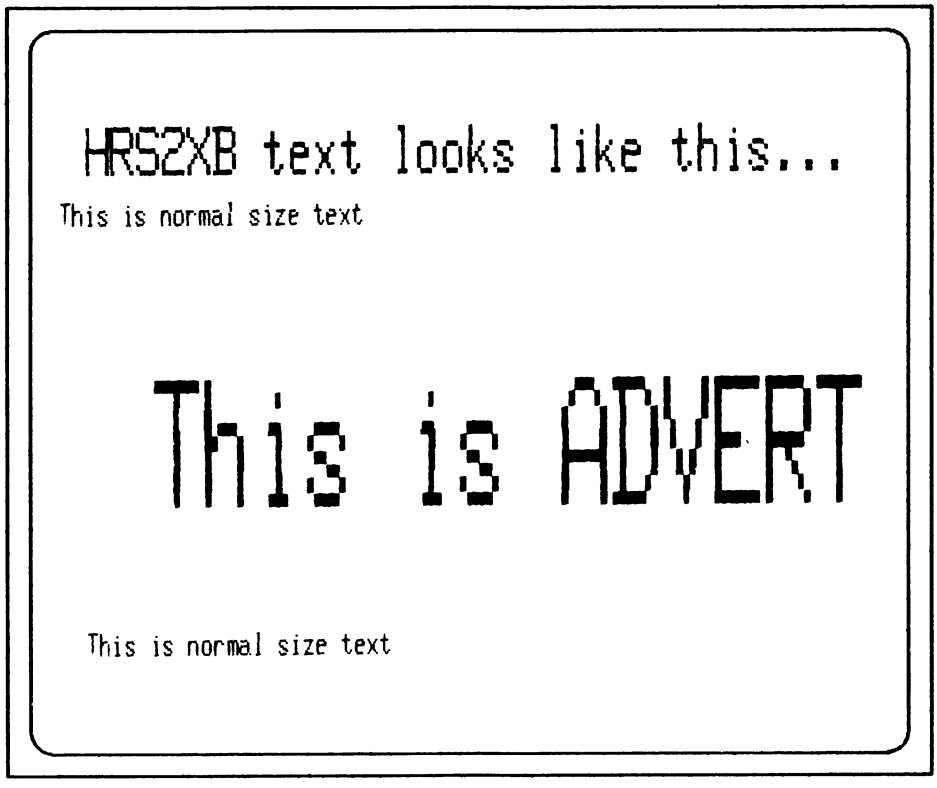
```
00620          OUT     (ØBH),A ; Reset ROM read latch
00630 NXTBIT   POP     AF      ; Restore ROM character
00640          PUSH    AF
00650          AND     C       ; Bit mask
00660          JR      Z,NOBIT ; Bit ON?
00670          CALL    8Ø3Ø
00680          INC     HL
00690          CALL    8Ø3Ø    ; Set 4 HIRES points
00700          DEC     DE
00710          CALL    8Ø3Ø
00720          DEC     HL
00730          CALL    8Ø3Ø
00740          INC     DE
00750 NOBIT    INC     HL      ; Increment X coord. for next bit
00760          INC     HL
00770          SRL     C       ; Shift bit mask
00780          JR      NC,NXTBIT
00790          POP     AF      ; Cut back stack
00800          DEC     DE      ; Decrement Y coord for next char
00810          DEC     DE
00820          POP     HL      ; Restore X coord.
00830          INC     IX      ; Next ROM address
00840          DJNZ    NXTBYT
00850          LD      HL,(CRSLOC)     ; Get cursor address
00860          LD      A,3FH
00870          AND     L
00880          CP      3DH     ; Check for end of line
00890          JR      C,SAMELN
00900          LD      DE,ØØ4Ø ; Skip a line
00910          ADD     HL,DE
00920 SAMELN   INC     HL      ; Incr. 2 char positions
00930          INC     HL
00940          LD      (CRSLOC),HL     ; Update cursor address
00950          RET
00960          END
```

The sample BASIC program below demonstrates the use of this subroutine to display double size characters on the screen as they are typed on the keyboard. You will notice that it is considerably faster than the earlier program which set the HIRES points in BASIC.

A slightly different technique is used to store the machine code in this program due to the sometimes unpredictable results of LISTing machine code which has been POKEd into REM statements. Memory space is reserved by DIMensioning array M(62), and the location of this memory area is calculated from the pointer read from the appropriate entry in the integer table. The machine code is then POKEd into this area in the usual way. When you RUN the program, the screen will go blank as line ØØ11Ø is executed, and then any character you type will be echoed to the screen almost instantly, until the PCG characters are exhausted.

Program name : 2XHRSB

```
00001 DIM M(62): REM     Array to contain m/l subroutine
00002 A = PEEK(1736)+256*PEEK(1737)+4: REM  A contains memory
      address  of array M into which m/l will be poked
00003 FOR I=0 TO 123: READ H: POKE A+I,H: NEXT I: REM  load m/l
00010 DATA  42, 11,  1, 62, 63,165, 22,  0, 95,175
00011 DATA 203, 19,203, 19,203, 19,203, 18,213, 62
00012 DATA 192,165,203, 63,203, 63, 95, 62,  3,164
00013 DATA  15, 15,179, 47, 95, 22,  0, 46,  0, 97
00014 DATA   6,  4, 55,203, 28,203, 29, 16,249,229
00015 DATA 221,225,225,  6, 16,229, 14,128, 62,  1
00016 DATA 211, 11,221,126,  0,245,175,211, 11,241
00017 DATA 245,161, 40, 16,205, 48,128, 35,205, 48
00018 DATA 128, 27,205, 48,128, 43,205, 48,128, 19
00019 DATA  35, 35,203, 57, 48,229,241, 27, 27,225
00020 DATA 221, 35, 16,207, 42, 11,  1, 62, 63,165
00021 DATA 254, 61, 56,  4, 17, 64,  0, 25, 35, 35
00022 DATA  34, 11,  1,201
00100 REM This is where the program really starts, all before this
      point is to load machine language subroutine.
00110 HIRES:CURS 0
00120 A0$=KEY:IF A0$="" THEN 120
00130 B=USR(A,ASC(A0$)):REM Call driver routine
00135 IF USED >123 THEN INPUT"PCG Characters exhausted";A1$:GOTO110
00140 GOTO 120
00150 END
```



AUDIBLE TACTILE FEEDBACK

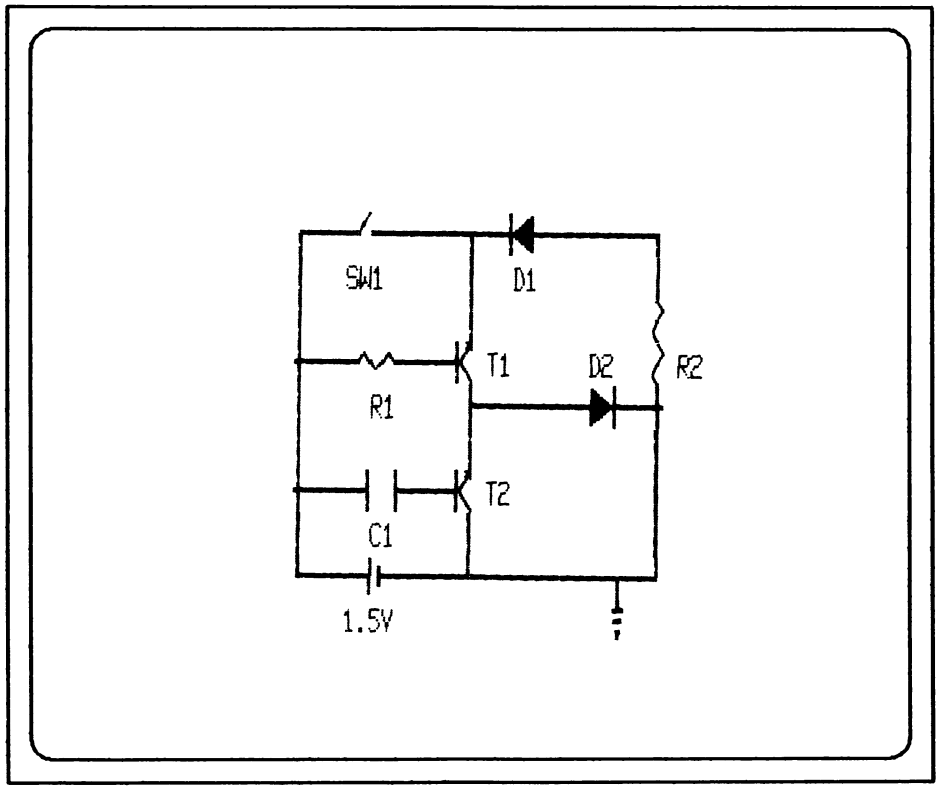
The above title is the technical way of saying that a keyboard will respond with a clicking sound each time a key is pressed. It is found on many expensive computers and terminals, and is designed to inform the operator that a keypress has been registered by the computer.

Program name : KEYCLK

```
00100 REM              ' KEYCLICK '                    Mark Bishop
00110 REM
00120 REM THIS PROGRAM PRODUCES AN AUDIBLE CLICK  WHEN A KEY IS
00130 REM PRESSED, VERY USEFUL IF YOU'RE NOT A TOUCH TYPIST AND
00140 REM TYPING IN A LISTING, OR IF YOUR 'BEE HAS STICKY KEYS.
00150 REM THE 'CLICK' TONE CAN BE CHANGED, BY CHANGING THE DATA
00160 REM ITEMS  IN LINE 290 TO  0,  AND  IN LINE 310  TO 6317.
00170 REM    RUN 'KEYCLICK' THEN ENTER 'NEW' AND START TYPING.
00180 REM      TO DISABLE 'KEYCLICK'  COLD-START YOUR 'BEE.
00190 REM
00200 FOR X=12288 TO 12354: READ D: T=T+D: POKE X,D: NEXTX
00210 READ C: IF T<>C THEN PRINT "DATA ENTRY ERROR": END
00220 I=USR(12288)
00230 END
00240 DATA  42,194,  0, 34, 37, 48, 42,160,  0, 17,  0,  2,183
00250 DATA 237, 82, 34,160,  0,229,235, 33, 35, 48,  1, 32,  0
00260 DATA 237,176,225, 34,194,  0,195, 33,128,229,205,233,163
00270 DATA  32, 24,245,197, 14,  5, 62, 64,211,  2,  6,  0, 16
00280 DATA 254,175,211,  2
00290 DATA   6,  0, 16,254, 13, 32,238
00300 DATA 193,241,225,201
00310 DATA 6876
```

ELECTRONIC SYMBOLS

The following program serves as another illustration of the programmable character graphics (PCG) capability of the MicroBee. The letters a to t have been redefined to represent electronic circuit symbols. You may define some more to suit your program using the technique described in Wildcards Volume One. The illustrated circuit is entirely imaginary, and should not work. Characters that are repeated in a line have not been entered in the more efficient 'square brackets' format in order to make the circuit interconnections easier to visualise.



Program name : SYMBOL

```
00100 CLS
00110 FOR I = 65040 TO 65439 :REM PCG poke addresses for a to y
00120 READ D
00130 POKE I,D
00140 NEXT I
00150 PCG: REM The circuit
00160 PRINT"oddddxddddddddduddabcddddddddddj"
00170 PRINT"errrrrrrrrrrrvrrrrrrrrrrrrrre"
00180 PRINT"errrrrrrrrrrrvrrrrrrrrrrrrrrm"
00190 PRINT"fddddklkddddaqrrrrrrrrrrrrrrm"
00200 PRINT"errrrrrrrrrrvddddddddddhigddf"
00210 PRINT"errrrrrrrrrrvrrrrrrrrrrrrrre"
00220 PRINT"fddddarrgdddaqrrrrrrrrrrrrrre"
00230 PRINT"errrrrrrrrrrvrrrrrrrrrrrrrre"
00240 PRINT"pdddddydddddwddddddddddsddn"
00250 PRINT"rrrrrrrrrrrrrrrrrrrrrrrtrrr"
00260 NORMAL  : REM circuit text
00270 CURS  68: PRINT"SW1"
00280 CURS  81: PRINT"D1"
00290 CURS 262: PRINT"R1"
00300 CURS 207: PRINT"T1"
00310 CURS 215: PRINT"D2"
00320 CURS 222: PRINT"R2"
00330 CURS 399: PRINT"T2"
00340 CURS 454: PRINT"C1"
```

```
00350 CURS 580: PRINT"1.5V"
00360 END
00370 REM  Characters are a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t
00380 DATA 3,3,3,3,3,3,3,255,255,3,3,3,3,3,3,3
00390 DATA 0,0,0,0,3,15,63,255,255,63,15,3,0,0,0,0
00400 DATA 0,12,60,252,252,252,252,255,255,252,252,252,252,60,12,0
00410 DATA 0,0,0,0,0,0,0,255,255,0,0,0,0,0,0,0
00420 DATA 24,24,24,24,24,24,24,24,24,24,24,24,24,24,24,24
00430 DATA 24,24,24,24,24,24,24,255,255,24,24,24,24,24,24,24
00440 DATA 192,192,192,192,192,192,192,255,255,192,192,192,192,
      192,192,192
00450 DATA 0,48,60,63,63,63,63,255,255,63,63,63,63,60,48,0
00460 DATA 0,0,0,0,192,240,252,255,255,252,240,192,0,0,0,0
00470 DATA 0,0,0,0,0,0,0,248,248,24,24,24,24,24,24,24
00480 DATA 0,0,0,0,24,60,102,195,129,0,0,0,0,0,0,0
00490 DATA 0,0,0,0,0,0,0,129,195,102,24,0,0,0,0,0
00500 DATA 24,12,6,3,3,6,12,24,24,48,96,192,192,96,48,24
00510 DATA 24,24,24,24,24,24,24,248,248,0,0,0,0,0,0,0
00520 DATA 0,0,0,0,0,0,0,31,31,24,24,24,24,24,24,24
00530 DATA 24,24,24,24,24,24,24,31,31,0,0,0,0,0,0,0
00540 DATA 15,7,15,13,24,48,96,192,192,96,48,24,12,6,3,1
00550 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
00560 DATA 0,0,0,0,0,0,0,255,255,24,24,24,24,24,24,24
00570 DATA 24,24,24,255,255,0,0,126,126,0,0,60,60,24,24,0
00580 DATA 0,0,0,0,0,0,0,255,255,3,3,3,3,3,3,3
00590 DATA 3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3
00600 DATA 3,3,3,3,3,3,3,255,255,0,0,0,0,0,0,0
00610 DATA 3,6,12,24,48,96,192,128,128,0,0,0,0,0,0,0
00620 DATA 0,192,192,192,195,195,195,195,195,195,195,195,192,192,
      192,0
```

TIPS AND TECHNIQUES

DISK TIPS

This part of this chapter does not pretend to be a complete guide to operating your Disk MicroBee: you cannot avoid reading the Disk System Manual and some standard texts on CP/M such as CP/M AND THE PERSONAL COMPUTER by Dwyer & Critchfield, INSIDE CP/M by David E Cortesi etc. We will try to show you some shortcuts and tips to make your transition to 'real' computing much easier. Certain statements do not apply to the 128k model: please check 128k manual before reading the next few paragraphs.

GETTING STARTED WITH DISKS

You should DDFORMAT and SYSGEN your blank disks as per the manual. You then must CP (or PIP) onto each fresh disk a copy of your most important utilities. You will need STAT.COM and a copy utility such as PIP.COM or CP.COM. STAT is useful amongst other things for determining the free space left on the disk. A copying utility is essential for transferring files between disks.

Expect to do a considerable amount of disk copying in the first few weeks, as there are literally hundreds of public domain programs that you may feel like having. Unless you have obliging friends with a collection of such programs, you will have to purchase them from a dealer or a user group (at a very nominal handling cost), or use a modem to download from a remote CP/M system (RCPM). Most large cities tend to have at least one RCPM. It may be better to ask an experienced disk system user what public domain utilities are 'worth having', or else you will waste a lot of time and effort.

It is worth cataloguing your collection of disks with the aid of a suitable commercial or public domain utility (which is usually a set of related programs). A cataloguing program keeps a record of every disk you process through it and the contents of each disk. You should have a unique disk number on each disk, such as '!N.ØØ1', '-CAT.ØØ1' etc. This is achieved by entering:

SAVE 1 !N.ØØ1

You could do a SAVE Ø, but we will soon tell you why a SAVE 1 is safer. Disks should be numbered in sequence. An accompanying utility such as FIND.COM allows you to find a particular program by displaying the number of each disk the desired program happens to be on. (You can do an ambiguous filename search too). This will be an excellent way of knowing what is in your collection and

avoid unwanted duplication. You may have noticed that the DIRectory shows your programs in the order in which you saved them (unless you ERAsed a program and subsequently inserted a smaller program). You may prefer to see the DIRectory sorted alphabetically. There are many public domain programs that do this for you, and we prefer one called 'XDIR.COM'. A more permanent way to achieve this is to use another program called SAP.COM (Sort And Pack). Public Domain programs usually have many versions in existence, so try and get the latest one. Some versions of SAP not only sort and rewrite the directory, but delete dummy files that have a length of zero. Hence make all dummy files length 1, unless you have the latest version of SAP. The reason we put an exclamation mark before the disk number is that it is the lowest value ASCII printable character, and SAP will always sort it to the start of the Directory. So a typical fresh disk may present the following type of DIRectory:

```
!N       .Ø34 | CP       .COM | STAT    .COM
```

CONVERTING *.MWB FILES TO ASCII FILES

If you are a prolific author or editor of a newsletter, or simply want to format your listings under WordStar, this tip will be useful. Readers of Wildcards Volume One will recall a program called ASCLST which allowed you to merge Microworld BASIC programs with Word-Bee files. A similar program is at the end of this chapter. The method of merging BASIC with WordStar is even easier - you are not even constrained by Word-Bee's refusal to accept 'words' >64 characters, as a packed DATA line might appear. WordStar will therefore accept long lines that contain no spaces. Here is the procedure:

1. Boot disk and type BASIC <CR> (to load BASIC)
2. Type LOAD "filename" <CR> (name of your BASIC program)
3. Type CLOSE2:OPEN"O",2,"DUMMY.ASC":OUT#2:LIST:OUT#Ø:CLOSE2 <CR>

NOTE: In step 3 above, it <u>is</u> the letter 'O', not zero! "DUMMY.ASC" (or any filename and extension of your choice) is the resultant ASCII file which can be read by and merged with WordStar.

PAGINATING *.DOC FILES ON A PRINTER

One way to get a hardcopy of an ASCII file (*.DOC or .TXT etc) is to press ^P before issuing the command TYPE xxxxxxx.DOC. This will certainly print on paper, but you will not get a crease gap. To get a properly paginated printout, issue the command (you must have PIP.COM in one of the drives):

```
PIP LST:=FILENAME.EXT[FP56]
```

The 'F' deletes formfeeds, and 'P56' starts a new page every 56 lines by sending a form feed command to the LST device.

CUSTOMISING WORDSTAR Version 3.Ø

by Richard C. Gration

WordStar 3.Ø as supplied with some early disk MicroBees was not accompanied by INSTALL.COM, which is a program that would have permitted you to make minor, (but menu-driven) changes to WS.COM such as setting default values on Help Level, Insert Toggle, Omit Page Number etc. Do not despair, as you do have a copy of DDT.COM on your system master disk and you can modify several areas without INSTALL. These instructions describe the steps that should be followed to adapt WordStar 3.Ø for use with dot matrix printers that require special control sequences or escape codes to provide special effects. All the steps are done using DDT.COM to patch the appropriate code into WS.COM.

NOTES: 1. Use a copy of your WS.COM, not the master!
2. WordStar 3.3 is supplied with Install.Com.

ALTERNATE CHARACTER SET

Most dot matrix printers support a second character set, which is usually an italic font. The ^PA command of WordStar 3.Ø (alternate character set) can be adapted to enable the italic font as follows:

a. Type 'DDT WS.COM' to load DDT and WS.COM into memory and begin editing:

```
ddt ws.com                  ;load WS.COM
DDT VERS 2.2                ;ddt signs on
NEXT PC
3FØØ ØlØØ                   ;file length and start
```

b. You must now select the codes that you wish to send to your particular printer. For example, the '8Ø' series of printers uses 'ESC "4"' (1B 34 hex) to switch on the italic character set.

c. The format required by WordStar 3.Ø for user patches is as follows:

NN - total number of bytes in control sequence (1 byte)

AA BB CC DD - actual bytes that are to be sent to the printer (up to a maximum of 4 bytes)

Thus for the '80' series of printers the code to be patched would be:

Ø2 - two bytes of data to be sent

1B 34 - control code to enable italic font

d. Type 'S6B5' in response to the '-' prompt of DDT. This will enable the user to 'Set' or alter memory at the specified location in the temporary working file that DDT sets up. Ø6B5 hex is the location of the code used for ^PA.

e. Enter these values into the file (user keystrokes are shown underlined) as follows:

```
-S6B5
Ø6B5 ØØ Ø2
Ø6B6 ØØ 1B
Ø6B7 ØØ 34
Ø6B8 ØØ .   (The period is used to tell DDT
             that alterations are complete)
```

f. The alterations for ^PA are now complete.

NORMAL CHARACTER FONT

The user must be able to reselect the normal character font after having used the italic font so the appropriate code must be patched into WS.COM. The method is exactly the same as above, however this time the code must be patched into locations Ø6BA to Ø6BE. Using the '80' series printer as an example again, the required code is:

Ø2 - number of bytes in control code

1B 35 - control code for italic font cancel

Using the same method as for the alternate character set this code should now be patched into WS.COM.

USER PATCHES

WordStar 3.Ø also provides 4 user definable codes (^PQ, ^PW, ^PE, and ^PR) that can be used to produce special effects. The locations that hold these codes are:

```
Ø6C9 - Ø6CD    user patch 1 (^PQ)
Ø6CE - Ø6D2    user patch 2 (^PW)
Ø6D3 - Ø6D7    user patch 3 (^PE)
Ø6D8 - Ø6DC    user patch 4 (^PR)
```

The format of the data in these locations is exactly the same as for the alternate character set, ie. the first byte is a count of the number of bytes of data to be sent (up to a maximum of 4 bytes) and the next 4 bytes contain these codes.

For example, to define ^PQ so that it will disable the paper out sensor on an '8Ø' type printer, enter the following data:

```
-SØ6C9
Ø6C9 ØØ Ø2
Ø6CA ØØ 1B  }
Ø6CB ØØ 38  } code to ignore 'paper out' sensor
Ø6CC ØØ .
```

DEFAULT HELP LEVEL

The MicroBee release version of WordStar 3.Ø has a default to level Ø. i.e. no help at all unless you select H at the NO FILE MENU every time you use WS.COM and set the level to 3 for beginners, or 2 for intermediate users, and so on. This can be cured with a patch through DDT.COM.

```
-s36Ø
Ø36Ø ØØ Ø3  } help level 3
Ø361 ØØ FF
Ø362 FF .
```

If you want a default help level of 2, substitute 2 instead of 3 at address 36Ø.

SAVING THE REVISED FILE

Once all the changes have been made to the temporary copy via DDT, the file must be saved back to the disk. To do this type 'gØ' or ^C in response to the DDT prompt '-'. This will return the user to CP/M and allow the file to be saved using the normal CP/M command 'save'. Type 'SAVE 62 filename.ext' to save the new file, where filename.ext is the name that the user wants. The new file may be saved as 'WS.COM', however as this will overwrite the original version, you should have another copy elsewhere in case an error has been made or you wish to return to the original version.

DISK BASIC TIPS

1. In a twin disk system, under BASIC.COM, you may view the DIRectory of drive B by typing:

```
DIR "B:*.*"
```

   This obviates the requirement to have BASIC.COM on each disk.

2. If you are using TBASICC.COM and wish to return to CP/M, you cannot do so by pressing RESET, as you will merely get the BASIC 'Ready' prompt. You need to press B and RESET simultaneously to return to CP/M.

ROUNDING REAL NUMBERS

The following routine illustrates how real numbers can be rounded to as many decimal places as required when printing values. The number of decimal places will be reflected in the value of the variable C1 and the print format statement. For Ø decimal places, C1 will be .5, for 1 decimal place it will be .Ø5 and so on. The true value of the variable A1 is unchanged, as it may be required for subsequent calculations:

Routine name : ROUNDING

```
ØØ1ØØ REM routine for rounding Real Numbers
ØØ11Ø CLS
ØØ12Ø INPUT "Enter sample Real Number - ";A1
ØØ13Ø C1 = .ØØ5 : REM 2 decimal places
ØØ14Ø PRINT [F8.2 A1+C1] : REM 2 decimal places
```

RANDOM NUMBER SELECTION USING STRING MANIPULATION

by David Dundas

In many BASIC programs, especially games, random selection from a list (usually DATA) is required. This is easily achieved with the line:

```
ØØ1ØØ X=INT(RND*ZØ)+1:   rem ZØ may equal any number
```

Now, this procedure is perfect if the inevitable repetition of a number does not ruin the object of the program. If it does, then the previously selected number must not be used again. The obvious method to check if a number has been used, is to relabel

each choice and loop the new selection through an 'IF...THEN' routine and send it back until it picks a previously unused number.

Routine name : RANSTG1

```
ØØ1ØØ A=A+1: REM A is the choice counter...Initialised A=Ø
ØØ11Ø X=INT(RND*ZØ)+1
ØØ12Ø IF A=1 THEN 16Ø
ØØ13Ø FOR B=1 TO A
ØØ14Ø IF X=P(B) THEN NEXT * B 11Ø
ØØ15Ø NEXT B
ØØ16Ø P(A)=X
```

Unfortunately, this re-checking method becomes more and more time-consuming as the number of choices diminish, and is only viable if the list consists of less than 1Ø elements. So, an alternative method must be used where larger lists are concerned.

To save processing time, the number of elements (ZØ) should be decremented by 1 (ZØ=ZØ-1) after each choice, while also keeping track of which members of the list have been already used. This is achieved with the aid of 'String Manipulation' e.g.

Routine name : RANSTG2

```
ØØ1ØØ ZØ=FLT(Z)           : REM Number in list
ØØ11Ø MØ$="": NØ$=""      : REM creating two empty strings
ØØ12Ø FOR X=1 TO Z+1 : MØ$=MØ$+CHR(X+31): NEXT X
ØØ13Ø S=INT(RND*ZØ)+1 : ZØ=ZØ-1
ØØ14Ø NØ$=MØ$(;S,S) : MØ$=MØ$(;1,S-1)+MØ$(;S+1)
ØØ15Ø G=ASC(NØ$)-32       : REM g is the selected random number
```

A string of ASCII characters is created (MØ$), having the same number of characters as the number of elements in the list plus 1 (the extra character is needed, or else line 14Ø would crash). The characters used are from ASCII #32 (space) to ASCII #126 (~): this is so we do not use the Z8Ø instruction codes at ASCII #Ø to ASCII #31, which will cause the program to abort; moreover, the string can be printed to check that the routine is working properly. In order to use more than these 96 choices in a program, additional strings can be created with the same ASCII characters, but with the label MØ$(1), MØ$(2) etc.

MULTIPLE FUNCTIONS FOR JOYSTICK 'FIRE' BUTTON

by David Dundas

Any switch connected to a computer can be used to toggle between two functions. This short routine can give a switch (for example, the 'Fire' button on the joystick) an extra function by detecting whether the button is being depressed for a given length of time.

Routine name : JOYFIRE

```
00100 B=IN(0): REM b=127 when fire button pressed
00110 IF B=127 THEN B ELSE 100
00127 PLAY 24;0,4: IF IN(0)<240 THEN LET F=1: PLAY 1;0,2:GOTO100
00128 IF F=3 OR F=1 THEN LET F=2: GOTO 100
00129 F=3: GOTO 100
```

(The variable F may be replaced by a line number or any other statement e.g. GOSUB) The PLAY statements are used both as an audible signal that a function has been selected, and as a means to standardise the length of time the button must be depressed on either a 2 MHz or 3.375 MHz MicroBee.

DID YOU KNOW THAT.....?

1. A space is not required in the syntax of print formatting of floating point numbers.

   i.e. [F8.2D1] is as good as [F8.2 D1]

2. You can insert a CONTROL G (^G) after REMs and between quotations in PRINT and DATA lines simply by pressing the CTRL and G keys simultaneously at the appropriate place. You will not see anything appear in the program except when you EDIT the relevant line - it will appear as the underline character. Using this method to cause a warning 'beep' is easier than the statement PLAY 23, and saves memory. It is usually found in PRINT statements that display a warning prompt message.

3. DATA lines may not contain multiple statements, e.g. a REM on the same line, or else an error will occur.

MERGING BASIC OR EDASM FILES WITH WORD-BEE

by Robin Clipsham

It can often be useful to load a program listing as a Word-Bee file so that it may be edited and have additional text added. Anyone who writes an article for a magazine or book, or merely wishes to print his programs with headers and page numbers on each page will appreciate the value of this facility. Compare this program with the one entitled ASCLST in Wildcards Volume One.

The following program allows a cassette file which has been created using redirected printout, to be loaded or appended to the end of an existing Word-Bee file. Both BASIC and Assembler versions provided here may be loaded at any convenient location, F400 Hex usually being fairly safe.

Program name : WBLODS

```
00100           DEFR    16D
00110           ORG     0F400           ;(or wherever)
00120           LD      A,(0500)
00130           CP      55
00140           RET     NZ              ;Word-Bee not initialised
00150           LD      A,08
00160           LD      (00E4),A        ;Select 1200 Bd cass in
00170           LD      HL,EXIT
00180           LD      (00A2),HL       ;Go exit on Reset
00190           LD      HL,(051DH)      ;WB end of file pointer
00200           XOR     A
00210 DECR      OR      (HL)
00220           JR      NZ,READY
00230           DEC     HL              ;Reduce until first non zero
00240           JR      DECR
00250 READY     INC     HL
00260 LOOP      CALL    803FH           ;Read cassette byte
00270           OR      A
00280           JR      Z,LOOP          ;Drop 00
00290           LD      (HL),A
00300           INC     HL
00310           LD      (051DH),HL
00320           LD      B,A
00330           CALL    800CH
00340           JR      LOOP
00350 EXIT      LD      A,01
00360           LD      (00E4),A        ;Restore k/b input
00370           JP      0C000
00380           END
```

WORD-BEE SOURCE LOADER (BASIC Version)

```
00100 FOR I = 0 TO 52:READ H:POKE 62464+I,H: NEXT I
64001 DATA  58,  0,  5,254, 85,192, 62,  8, 50,228
64002 DATA   0, 33, 45,244, 34,162,  0, 42, 29,  5
```

```
64003 DATA 175,182, 32,  3, 43, 24,250, 35,205, 63
64004 DATA 128,183, 40,250,119, 35, 34, 29,  5, 71
64005 DATA 205, 12,128, 24,239, 62,  1, 50,228,  0
64006 DATA 195,  0,192,  0,  0,  0,  0,  0,  0,  0
```

OPERATION

1) Assemble the above source code or enter the BASIC program as required, and save as an 'M' type file using monitor. (Described on page 13 of Wildcards Volume One)

2) Load program which you wish to convert to Word-Bee format.

3) Create cassette file as follows:

   a) For BASIC program start cassette and type:
      OUTL #3:LLIST <CR>

   b) For EDASM program, go to Monitor and Alter the contents of the List Output Device Byte at 00E3 to 08. Return to EDASM, start cassette and list program using L command or Assemble using A/L command

4) Initialise Word-Bee then go to monitor and load the File Loader program from cassette. Go to F400 and start cassette. You will see lines displayed as they are loaded into the Word-Bee file. When the cassette has finished loading, press RESET to get back to Word-Bee.

LOGICAL OPERATORS

In MicroWorld BASIC, the logical operators are represented by the three functions AND, OR and NOT. They are sometimes called the Boolean logic functions, because they perform the 'truth-table' operations in a bitwise fashion (on each bit of the bytes as arguments). In this version of BASIC, the general form of their use is:

```
00100 X = (A AND B)
00200 X = (A OR B)
00300 X = (NOT A)
```

Note that each expression must be in parenthesis (enclosed in brackets) and that two variables are used in the AND and OR operations, while the NOT function uses only one.

The outcome of a logical operation (on each bit of the bytes) is determined as follows:

NOT

A	(NOT A)
1	Ø
Ø	1

AND

A	B	(A AND B)
1	1	1
1	Ø	Ø
Ø	1	Ø
Ø	Ø	Ø

OR

A	B	(A OR B)
1	1	1
1	Ø	1
Ø	1	1
Ø	Ø	Ø

Thus the NOT function simply reverses all the bits in the byte being addressed, the AND function yields a '1' (bit 'on') only when both corresponding bits are a '1', while the OR function yields a '1' if either bit is a '1'. Note particularly that the OR function also yields a '1' when both bits are on. Thus it is not an 'exclusive' OR, which is represented in some versions of BASIC as XOR. If you have created BASIC programs of your own, this explanation should not be too difficult to follow, as the same logic applies to a line in BASIC such as:

```
ØØ35Ø IF A>3 OR A<1Ø THEN 38Ø
```

In this instance, the program would continue execution at Line 38Ø if both A>3 and B<1Ø were true, as well as when one or the other was true.

If you are new to programming and are already finding difficulty in absorbing the logic, do not despair - the next section will give you an opportunity to gain experience in manipulating these 'bits and bytes' to your heart's content.

A DEMONSTRATION PROGRAM

Our next step is to create and RUN a program which will allow you to INPUT values for A and B, apply the functions AND, OR or NOT to them at will and observe on your VDU the fate of each bit and the yielded value of each expression.

In addition, the program will 'test' the result of the execution

of each function for 'True' (negative) or 'False' (not negative).

Program name : AND-OR

```
00100 REM ---- Program to demonstrate use of AND, OR, NOT -----
00110 CLS
00120 GOTO 1020 : REM ---- Display menu

00130 REM ---- Subroutine for Use of AND ----------------------
00140 GOSUB 540 : REM ---- Re-display title
00150 UNDERLINE : PRINT "Use of AND :" : NORMAL
00160 GOSUB 730 : REM ---- Input Values for A & B
00170 X=(A AND B)
00180 UNDERLINE : PRINT\ "X=(A AND B)" : NORMAL
00190 PRINT "PRINT X :";
00200 PRINT X; " (Prints ";X;")"
00210 GOSUB 820 : REM ---- Display binary format
00220 GOSUB 680 : REM ---- Next screen
00230 GOSUB 930 : REM ---- Test for X
00240 GOSUB 680
00250 GOTO 1020

00260 REM ---- Subroutine for Use of OR -----------------------
00270 GOSUB 540
00280 UNDERLINE : PRINT "Use of OR :" : NORMAL
00290 G=-1 : GOSUB 730
00300 UNDERLINE : PRINT\ "X=(A OR B)" : NORMAL
00310 X=(A OR B)
00320 PRINT "PRINT X :";
00330 PRINT X;" (Prints ";X;")"
00340 GOSUB 820
00350 GOSUB 680
00360 GOSUB 930 : REM ---- Test for X
00370 GOSUB 680
00380 GOTO 1020

00390 REM ---- Subroutine for Use of NOT ----------------------
00400 GOSUB 540
00410 UNDERLINE : PRINT "Use of NOT :" : NORMAL
00420 H=-1 : GOSUB 730
00430 UNDERLINE : PRINT\ "X=(NOT A)" : NORMAL
00440 X=(NOT A)
00450 PRINT "PRINT X :";
00460 PRINT X;
00470 PRINT " (Prints ";X;")"
00480 GOSUB 820
00490 GOSUB 680
00500 GOSUB 930 : REM ---- Test for X
00510 GOSUB 680
00520 GOTO 1020
```

```
00530 REM ---- Subroutine to display title --------------------
00540 CURS 18 : UNDERLINE
00550 PRINT "Development of Boolean Logic" : NORMAL
00560 RETURN
00570 REM ---- Subroutine to convert decimal to binary --------
00580 IF C : C=C+32768 : F=-1
00590 FOR J=1 TO 16
00600   A1=FLT(C/2) : A2=FLT(C)/2
00610   P=P-2 : CURS P
00620   IF F AND J=16 : PRINT " 1" : F=0 : RETURN
00630   IF A1<>A2 : PRINT " 1";ELSE PRINT " 0";
00640   C=C/2
00650 NEXT J
00660 RETURN

00670 REM ---- Subroutine to move to next screen --------------
00680 PRINT\ "     Press any key to continue";
00690 A0$=KEY : IF A0$="" THEN 690
00700 CLS
00710 RETURN

00720 REM ---- Subroutine to input values for A & B -----------
00730 CURS 128 : INPUT "Value for A :";A;
00740 CURS 156
00750 PRINT "A =";A
00760 IF H : RETURN
00770 INPUT "Value for B :";B;
00780 CURS 220
00790 PRINT "B =";B
00800 RETURN

00810 REM ---- Subroutine to display binary format ------------
00820 P=546 : C=A : GOSUB 580
00830 CURS P+33 : PRINT " = ";A
00840 IF H : PRINT "NOT" : H=0 : GOTO 890
00850 IF G : PRINT "OR" ELSE PRINT "AND" : G=0
00860 P=P+160 : C=B : GOSUB 580
00870 CURS P+33 : PRINT " = ";B
00880 PRINT "   ";[A42 45]
00890 P=P+160 : C=X : GOSUB 580
00900 CURS P+33 : PRINT " = ";X
00910 RETURN

00920 REM ---- Subroutine to test for X ----------------------
00930 GOSUB 540
00940 UNDERLINE : PRINT\ "Test for X : (";X;")"
00950 PRINT\ "IF X : PRINT ''True''";
00960 PRINT " ELSE PRINT ''False''" : NORMAL
00970 IF X : PRINT\ "True"; ELSE PRINT\ "False";
00980 PRINT " (Prints ";
00990 IF X :PRINT"''True'' (-ve))"ELSE PRINT"''False'' (not -ve))"
01000 RETURN

01010 REM ---- Subroutine to display menu --------------------
01020 GOSUB 540 : REM ---- Display title
```

```
Ø1Ø25 G=Ø : H=Ø
Ø1Ø3Ø CURS 15Ø : PRINT "1. Use of AND"
Ø1Ø4Ø CURS 214 : PRINT "2. Use of OR"
Ø1Ø5Ø CURS 278 : PRINT "3. Use of NOT"
Ø1Ø6Ø CURS 342 : PRINT "4. Exit to Basic"
Ø1Ø7Ø CURS 465 : INPUT "Enter Selection (1 to 4) :";S
Ø1Ø8Ø IF S<1 OR S>4 THEN PLAY 16 : CURS 492 : PRINT "  ":
      GOTO 1Ø7Ø
Ø1Ø9Ø CLS
Ø11ØØ ON S GOTO 14Ø,27Ø,4ØØ,112Ø
Ø111Ø GOTO 1Ø2Ø
Ø112Ø END
```

You should find the listing is sufficiently documented with REMs for you to follow the main flow of the program, but a little more comment may be helpful:

1. Note in the subroutine to convert decimal to binary (lines 57Ø to 66Ø) that there is a test for a negative number ('IF C' in line 58Ø). When you RUN the program you will find that if you INPUT a value between 32767 and 65536 that this is automatically converted to a negative number, while the routine used to perform the conversion requires a positive number. As all negative numbers are indicated by the high bit (of the two byte sequence) being set to '1', it is necessary for the conversion to 'flag' that bit and print a '1' after converting to a positive number for the purposes of printing the binary values to screen.

2. The binary display follows the accepted convention of Bit Ø being written on the right hand side and the higher Bits being progressively written to the left. However, the inclusion of unused (zero) Bits to the left of the highest set Bit is unconventional, but hopefully helpful for a demonstration program.

Before RUNning the program, SAVE it to tape (or disk) as a safeguard. Try entering a range of numbers so that the display gives you a good understanding of what happens with each of the functions. Include negative as well as positive numbers. However, do not forget that BASIC will not accept an integer of a greater value than 65536!

While the purpose of the program is to enable you to gain familiarity with the functions under discussion, it can also be used simply to obtain the binary equivalents of a specific decimal number.

We will now move on to describing some of the specific applications which utilise the functions of AND, OR and NOT. However, keep this program handy to enable you to illustrate some of the examples that will be discussed.

APPLICATIONS

In WILDCARDS Volume 1 we included a technique for converting lower case input to upper case, by including a command 'POKE 257,1' in the BASIC program. However, to POKE memory locations is not always the best way of achieving a particular purpose. In this instance, breaking out of such a program, say deliberately with control-C, or unwittingly via an error message, will result in the computer converting all direct mode keyboard input to upper case until either a cold start is initiated or a further POKE command is given. Further, updated versions of BASIC may use a different scratch location for specific functions. Two simple subroutines can be used to convert to either upper or lower case as required, without resorting to the POKE command.

Program name : UCASE

```
ØØ1ØØ CLS
ØØ11Ø REM ---- Subroutine to convert lower to upper case ---
ØØ12Ø INPUT "Enter upper or lower case letter";AØ$
ØØ13Ø X=(ASC(AØ$) AND 95)
ØØ14Ø PRINT CHR(X);" (Always prints upper case)"
```

This little routine uses the AND function to strip off Bit 5 (the 6th Bit) to ensure that an upper case character is printed irrespective of whether a lower or upper case character has been entered from the keyboard. If you test out a few examples with the program AND-OR you will see why the decimal value of 95 achieves this objective. Note the use of the expression ASC(AØ$) to obtain the decimal value of the character.

The next routine uses the OR function to achieve a reverse result.

Program name : LCASE

```
ØØ1ØØ CLS
ØØ11Ø REM ---- Subroutine to convert upper to lower case ---
ØØ12Ø INPUT "Enter upper or lower case letter";AØ$
ØØ13Ø X=(ASC(AØ$) OR 32)
ØØ14Ø PRINT CHR(X);" (Always prints lower case)"
```

For this routine, we need to set Bit 5 on (change value to '1') if it is a zero, or leave it unaltered if it is already on (lower case character input). Check with the program AND-OR to see why this works if the value ORed is 32 decimal.

Further small routines can be developed using OR to produce INVERSE and UNDERLINE characters.

Program name : INVRSE

```
00100 CLS
00110 REM ---- Subroutine to inverse one character ----------
00120 INVERSE:NORMAL
00130 INPUT "Enter one character";A0$
00140 X=(ASC(A0$) OR 128)
00150 PRINT CHR(X);" (Always inverses the character)"
```

Program name : ULCHAR

```
00100 CLS
00110 REM ---- Subroutine to underline one character --------
00120 UNDERLINE:NORMAL
00130 INPUT "Enter one character";A0$
00140 X=(ASC(A0$) OR 128)
00150 PRINT CHR(X);" (Always underlines the character)"
```

Note that both procedures use the same value (128) to be ORed to change the input character to be either underlined or inversed. The only essential difference in the code for the routines is in Line 120. So the required mode has to be set once within each program where either of these routines is used.

The value 128 decimal when ORed with the value of the character input sets Bit 7 on, thus accessing the graphics mode characters.

The following program demonstrates why it is necessary to include Line 120 in the above routines.

Program name : UNDINV

```
00100 CLS
00110 REM ---- Subroutine to switch from underline to inverse
00120 UNDERLINE:NORMAL
00130 INPUT "Enter one character";A0$
00140 X=(ASC(A0$) OR 128)
00150 PRINT CHR(X);" (Always underlines the character)"
00160 PLAY 0,10;16
00170 INVERSE:NORMAL
00180 PRINT "Note the change to ''INVERSE'' after the ''pip''"
00190 CURS 67 : PRINT "Now changed to ''INVERSE'')       "
00200 END
```

While we are touching on graphics, here is a useful little trick to re-initialise PCG characters in a completed program to produce the inverse of the characters set by the DATA statements. It can also be helpful when developing graphics and you want to see the

effect of inversing the PCG without the tedium of preparing new DATA. Only one line needs to be added to produce the required effect. Bear in mind that this segment will not RUN on its own: it requires the full DATA statements and the PCG PRINT routines as well. The example given assumes a program which uses 21 PCG characters - it was taken from a demonstration program supplied by Applied Technology called FOMB.MWB.

Program name : INVPCG

```
00100 REM ---- Subroutine (part) to operate PCG with the
00110 REM      complements of the DATA statements
00120 FOR I = 64528 TO 64528+(16*21)-1
00130  READ D
00140  D=(NOT D) : REM ---- Converts DATA to its complement
00150  POKE I,D
00160 NEXT I
00170 REM DATA statements for 21 PCG characters included here
```

Line 140 is the only line that needs to be added to a completed program using PCG graphics. Of course, the line number will be different - it is placed between the READ and POKE statements used to initialise the PCG. You should have no difficulty in working out how it functions.

So we now have a range of examples of the use of logical operators: you should now be able to develop some ideas of your own. Before leaving this topic, however, we will introduce another aspect of what is termed 'bit coding'. If you have progressed through this section thoroughly, you will have a better understanding of what follows.

BIT CODING

You will have seen that each byte is represented by 8 bits and therefore that 16 bits requires two bytes. Integers in MicroWorld BASIC (and other versions operating with the Z80 series of microprocessors) are in the range -32768 to +32767. They are actually 16 bit, signed, two's complement integers. We have already seen that the 'sign' (whether the value is negative or positive) is indicated by whether the high bit (bit 15) is set or not. So each arrangement of 16 bits is represented by a unique number and so can be used as a code to store information in a program on whether a particular condition has been met or not (e.g. 'yes' or 'no', or 'on' or 'off').

So within a program, up to 16 'conditions' can be store in two bytes. This provides an enormous saving in memory where a large number of records are generated and stored by a program.

The following program demonstrates how such a bit-coding system can be developed.

Program name : RBIT16

```
00100 REM ---- Routine to save 16 replies in 2 bytes --------
00110 CLS
00120 CURS 19 : PRINT "---- Bit Coded Replies ----"
00130 A=0 : B=0 : F=0 : N=0 : Y=1
00140 FOR J=1 TO 16
00150   CURS J*32+160
00160   PRINT "Reply to Question #";J;" [Y/N]";
00170   INPUT A0$
00180   X=(ASC(A0$) AND 95)
00185   IF X=89 : A=A+Y ELSE LET A=A+N
00190   Y=Y*2
00200 NEXT J
00210 CURS 778
00220 PRINT "Decimal equivalent of the Binary Number=" A
00230 PLAY 0,30
00240 CLS
00250 IF A : B=A : A=A+32768 : F=-1 : GOSUB 360
00260 FOR J=1 TO 16
00270   CURS J*32+166
00280   A1=FLT(A/2) : A2=FLT(A)/2
00290   PRINT "Answer #" J " is ";
00300   IF F AND J=16 : PRINT "Yes" : END
00310   IF A1<>A2 : PRINT "Yes" ELSE PRINT "No"
00320   A=A/2
00330 NEXT J
00340 END

00350 REM ----- Subroutine to notify number conversion --------
00360 PLAY 16 : CURS 10
00370 PRINT "Binary number ";B;" changed to";A
00380 CURS 76
00390 PRINT " and high bit interpreted as 1"
00400 RETURN
```

As for the program AND-OR, you will see that all negative numbers are converted to positives for coding and decoding, with the high bit set to '1' if a negative number was generated in the first place. However, note that in an actual program you may not need to use all 16 bits. In that case, Lines 250, 300 and 350 to 380 will not be needed - but do not forget to change Lines 140 and 260 to reflect the revised number of bits used. Of course, if you are only using 8 bits (8 'conditions'), then the information is stored in only one byte.

SMALL BUSINESS SECTION

THE LOAN ARRANGER

Do you find it difficult to fathom the intricacies of such things as the Binomial Theorem these days? If you left school more than five years ago, then there is a distinct chance that you do. This is not surprising, as there are many things learnt at school that you will have little or no need for in the outside world.

This program will help you over some little used (long forgotten) arithmetic to work out the details of Mortgage repayments. Of course, these calculations could be done on an ordinary hand-held calculator if you remember your schooling. It would not, however, be as quick or convenient.

The program starts by asking for the amount, interest rate and term of the loan in months. It then supplies the necessary figure for monthly repayment. You then have the option of increasing the monthly payment and viewing the results as either a monthly analysis and totals, or just totals. If you have increased the monthly payment, then the printout of totals will show you in which month the last payment is made. Some surprises may be in store for you, when the analysis reveals how long it takes before the monthly capital redemption overtakes the monthly interest.

Program name : ARRANG

```
00100 REM*** LOAN ARRANGER ***
00110 REM*** by TONTO ***
00120 SD 14

00129 REM*** INPUT ***
00130 CLS:CURS 15
00140 INVERSE:PRINT" COMPOUND INTEREST LOAN CALCULATOR "
00150 NORMAL:PRINT
00160 INPUT "LOAN TOTAL        = $";LØ
00170 INPUT "INTEREST RATE     = %";RØ
00180 INPUT "TERM IN MONTHS    =  ";NØ
00190 LØ=LØ-FRACT(LØ)
00200 CLS:N=INT(NØ)
00210 PRINT "Loan of $";LØ;" to be repaid over";N;" months at";
00220 PRINT RØ;"%"\\\

00229 REM*** Calculate Monthly Payment ***
00230 MØ=RØ/1200
00240 KØ=(1+MØ)^NØ
00250 PØ=(LØ*MØ*KØ)/(KØ-1)
00260 P5=FRACT(PØ)
00270 PØ=(PØ-P5)+(P5*100+0.99)/100
00280 PRINT "Monthly payment is $";[F12.2 PØ]\\\
```

```
00289 REM*** OPTIONS ***
00290 PRINT "Do you wish to increase the monthly payment? (Y/N)";
00300 GOSUB 700
00310 IF Z7$="N" OR Z7$="n" THEN 350
00320 INPUT "ENTER NEW MONTHLY PAYMENT ";F1
00330 IF F1>P0:P0=F1:GOTO 350
00340 PRINT"MUST BE A LARGER PAYMENT":GOTO 320
00350 F1=P0
00360 PRINT"ENTER 'T' FOR TOTALS, OR 'A' FOR ANALYSIS"
00370 GOSUB 700
00380 IF Z7$="T" OR Z7$="t":Z7$="T":PRINT "BUSY.....";:GOTO 400

00389 REM*** Main Calculator ***
00390 GOSUB 640
00400 B0=L0*100:T1=0:T2=0:X=0:P0=P0*100
00410 FOR I=1 TO N
00420 T0=M0*B0:T0=T0+0.5
00430 T0=T0-FRACT(T0)
00440 IF I=N:P0=B0+T0
00450 T2=T2+P0:B0=B0-P0+T0
00460 IF B0<0:P0=P0+B0:T2=T2+B0:B0=0
00470 P1=B0/100:P2=T0/100
00480 T1=P0/100-P2
00490 IF Z7$<>"T" THEN 510
00500 IF B0=0 THEN NEXT* I 570 ELSE 560
00510 PRINT [I4 I];" ";[F12.2 T1];" ";[F12.2 P2];" "[F12.2 P1]
00520 IF B0=0:GOSUB 690:NEXT* I 570
00530 X=X+1:IF X<12 THEN 560
00540 GOSUB 690
00550 X=0:GOSUB 640:GOTO 560
00560 NEXT I

00569 REM*** Totals Output ***
00570 CLS
00580 PRINT  "MONTH NUMBER    =  ";I
00590 PRINT\\"MONTHLY PAYMENT = $";[F12.2 F1]
00600 PRINT  "FINAL PAYMENT   = $";[F12.2 (P0/100)]
00610 PRINT  "TOTAL PAYMENTS  = $";[F12.2 (T2/100)]
00620 GOSUB 690
00630 RUN

00639 REM*** Subroutines ***
00640 CLS:PRINT:INVERSE
00650 PRINT " MONTH   PRINCIPAL     INTEREST    BALANCE "
00660 NORMAL
00670 PRINT
00680 RETURN
00690 PRINT"PRESS ANY KEY TO CONTINUE";
00700 Z7$=KEY:IF Z7$="" THEN 700 ELSE PRINT:RETURN
```

Notice the use of 'RUN' in line number 630. This could be changed to:

```
00630 GOTO 130
```

If you do not know how long your mortgage still has to run at the present rates, enter the figure for outstanding principal, the current interest rate, and a term of 600 months (50 years). Now select the option for increased payment and enter your current monthly payment. Pressing 'T' for totals will now tell you how many monthly payments are left.

The results achieved with this program may vary slightly from those of your lending institution. Their number crunching machines may use more or less significant digits than we have done.

Sample RUN:
(note that such short simple loans are unlikely to be worked out in compound interest)

----------Screen 1

COMPOUND INTEREST LOAN CALCULATOR

LOAN TOTAL = $ 1200
INTEREST RATE = % 11.5
TERM IN MONTHS = 12

----------Screen 2

Loan of $ 1200. to be repaid over 12 months at 11.5%

Monthly payment is $ 106.34

Do you wish to increase the monthly payment? (Y/N) N

ENTER 'T' FOR TOTALS, OR 'A' FOR ANALYSIS A

----------Screen 3

MONTH	PRINCIPAL	INTEREST	BALANCE
1	94.84	11.50	1105.15
2	95.75	10.59	1009.39
3	96.67	9.67	912.71
4	97.59	8.75	815.12
5	98.53	7.81	716.58
6	99.47	6.87	617.10
7	100.43	5.91	516.66
8	101.39	4.95	415.27
9	102.36	3.98	312.90
10	103.34	3.00	209.55
11	104.33	2.01	105.21
12	105.21	1.01	0.00

PRESS ANY KEY TO CONTINUE

```
-------------------------------------------------Screen 4

MONTH NUMBER     =    12


MONTHLY PAYMENT = $    1Ø6.34
FINAL PAYMENT   = $    1Ø6.22
TOTAL PAYMENTS  = $  1276.Ø5

PRESS ANY KEY TO CONTINUE

------------------------------------------------- RUN
```

ANOTHER DATA BASE

An earlier version of a program named BEEDAT was sold in Melbourne and Canberra in early 1983. It met with reasonable success, but was neither advertised nor marketed in a way that might have made it available more widely. This version has been completely re-written in an easier to understand format, and the instructions which formed part of the original program have not been included in the code.

It should be noted that this program was not intended to simulate any commercially available data-base program. It is intended as a general purpose data handling tool, easily altered to suit many applications around the home and small business.

BEEDAT provides five text fields per record. The maximum number of records which can be stored depends on the size of records. Records are not set to any pre-determined length. All records are sorted upon insertion in field order. If necessary, all fields may be used in this insertion sort. Further sorting by a nominated field is also available.

NOTE that most REMs are actually called. This is to discourage those lazy people amongst us from ending up with a program that is so hard to follow that alterations are almost impossible. Be warned. (This is a deliberate deviation from normal convention.)

Program name: BEEDAT

```
ØØ1ØØ REM *** BEEDAT version 2.Ø ***
ØØ11Ø STRS (4ØØØ)
ØØ12Ø DIM DØ(1ØØ,5),D1(5)
ØØ13Ø N1$="NEW":C=96Ø:GOTO 2Ø4Ø

ØØ14Ø REM *** SORT ********************************************
ØØ15Ø IF R<2 THEN RETURN ELSE PRINT"FIELD #";:GOSUB 2Ø3Ø
ØØ16Ø GOSUB 2Ø1Ø:L=INT(VAL(Z7$))
```

```
00170 IF L<1 OR L>5:GOSUB 2140:RETURN
00180 PRINT"SORTING";:P=0:M=R
00190 M=M/2:IF M=0 THEN 290
00200 FOR K=1 TO M
00210   I=K:J=K+M:S=0
00220   IF D0$(I,L)<=D0$(J,L)THEN 260 ELSE LET S=1
00230   FOR N=1 TO 5
00240     D1$(N)=D0$(I,N):D0$(I,N)=D0$(J,N):D0$(J,N)=D1$(N)
00250   NEXT N
00260   I=J:J=J+M:IF J<=R THEN 220
00270   IF S<>0 THEN 210
00280 NEXT K:GOTO 190
00290 PLAY 10,2:RETURN

00300 REM *** LOAD FILE ****************************************
00310 R=0:CLS:GOSUB 2010:PRINT"LOADING...";
00320 IN#3:OUT#0 OFF
00330 INPUT Z7$:IF Z7$(;1,5)<>"*****"THEN 330
00340 N1$=Z7$(;11,16):R=INT(VAL(Z7$(;6,10))):POKE 61440+R,13
00350 FOR I=1 TO R:INPUT Z7$,D0$(I,1),D0$(I,2),D0$(I,3),D0$(I,4),
      D0$(I,5):IF INT(VAL(Z7$))<>I THEN NEXT*I 380
00360 POKE 61439+I,9:NEXT I
00370 IN#0:OUT#0:CLS:PRINTN1$:PLAY 9:RETURN
00380 R=I:IN#0:OUT#0:CLS:PRINT"BAD LOAD":PLAY 9,3;0,8:RETURN

00390 REM *** SAVE FILE ****************************************
00400 CLS:CURS C:INPUT"ENTER FILE NAME (6 char. max) ";N1$;
00410 N1$=N1$+"//////":N1$=N1$(;1,6)
00420 GOSUB 2010:PRINT"START TAPE AND PRESS <RET>";
00430 GOSUB 2030:PRINT"  RECORDING...";N1$;
00440 POKE 61440+R,13
00450 OUT#3:PRINT"*****";[I5 R];N1$
00460 FOR I=1 TO R:PRINT[I5 I];",";
00470 PRINT D0$(I,1);",";D0$(I,2);",";D0$(I,3);",";D0$(I,4);",";
      D0$(I,5)
00480 POKE 61439+I,9
00490 PLAY 0,4
00500 NEXT I
00510 PRINT"0,0,0,0,0,0,0,0,0"
00520 OUT#0:PLAY 9:RETURN

00530 REM *** INSERT ********************************************
00540 Q=0:FOR I=1 TO 5
00550   PRINT"FIELD #";I;" ";
00560   IF Q<134 THEN PRINT[A40 95]; ELSE PRINT[A14 95];
00570   CURS 969:INPUT""D1$(I);:GOSUB 2010:V=LEN(D1$(I))
00580   Q=Q+V:IF Q>170 OR V>40:Q=Q-V:GOSUB 2000:GOTO 550
00590 NEXT I
00600 GOSUB 2010:PRINT"SORTING";:R=R+1
00610 IF R>1 THEN 650
00620 FOR I=1 TO 5
00630   D0$(1,I)=D1$(I)
00640 NEXT I:RETURN
00650 FOR I=R-1 TO 1 STEP -1
00660   IF D0$(I,1)>D1$(1) THEN 720
```

```
00670   IF D0$(I,1)<D1$(1)THEN 690
00680   IF D0$(I,2)+D0$(I,3)+D0$(I,4)+D0$(I,5)>D1$(2)+D1$(3)+
        D1$(4)+D1$(5) THEN 720
00690   FOR J=1 TO 5
00700     D0$(I+1,J)=D1$(J)
00710   NEXT J:NEXT*I 770
00720   FOR J=1 TO 5
00730     D0$(I+1,J)=D0$(I,J)
00740     IF I=1:D0$(I,J)=D1$(J)
00750   NEXT J
00760 NEXT I
00770 RETURN

00780 REM *** DELETE ***********************************************
00790 INPUT"DELETE #"D;:IF D<1 OR D>R THEN GOSUB 2140:RETURN
00800 R=R-1:IF R<D THEN RETURN
00810 GOSUB 2010:PRINT"SORTING";
00820 FOR I=D TO R
00830   FOR J=1 TO 5
00840     D0$(I,J)=D0$(I+1,J)
00850   NEXT J
00860 NEXT I
00870 FOR I=1 TO 4:D0$(R+1,I)="":NEXT I:RETURN

00880 REM *** EDIT *************************************************
00890 INPUT"EDIT #"D;:IF D<1 OR D>R THEN GOSUB 2140:RETURN
00900 CLS:PRINT"#";D
00910 FOR I=1 TO 5
00920   PRINT TAB(5);I;" ";D0$(D,I)
00930 NEXT I:GOSUB 2010:PRINT"FIELD #";:GOSUB 2030
00940 F=INT(VAL(Z7$)):IF F<1 OR F>5 THEN GOSUB 2140:RETURN
00950 CURS 1,F+1:PRINT">>>":GOSUB 2010:PRINT"FIELD #";F;
00960 INPUT""D0$(D,F);:V=LEN(D0$(D,F))
00970 IF V>40:GOSUB 2000:RETURN
00980 CURS 8,F+1:PRINT[A45 32];:CURS 8,F+1:PRINT D0$(D,F);
00990 FOR I=1 TO 5
01000   D1$(I)=D0$(D,I)
01010 NEXT I:GOSUB 800:GOSUB 600
01020 RETURN

01030 REM *** FIND *************************************************
01040 PRINT"FIELD #";:GOSUB 2030:F=INT(VAL(Z7$)):GOSUB 2010
01050 IF F<1 OR F>5 THEN GOSUB 2140:RETURN
01060 INPUT"SEARCH FOR>> ";D1$(F);
01070 FOR I=1 TO R
01080   IF D0$(I,F)<>D1$(F) THEN 1140 ELSE CLS:PRINT"#";I
01090   FOR J=1 TO 5
01100     PRINT J;") ";D0$(I,J)
01110   NEXT J:GOSUB 2010
01120   PRINT"IS THIS THE ONE ? (Y/N) ";:GOSUB 2030
01130   IF Z7$="Y" THEN NEXT*I 1150
01140 NEXT I:CLS:PRINT\"SEARCH FINISHED":GOSUB 2010
01150 RETURN
```

```
01160 REM *** LIST ***********************************************
01170 CLS:PRINT:Z=0
01180 FOR I=1 TO R
01190   PRINT"#";I;
01200   FOR J=1 TO 5
01210     PRINT TAB(6);D0$(I,J)
01220   NEXT J:Z=Z+1
01230   IF Z=3:GOSUB 2020:Z=0:CLS:PRINT
01240 NEXT I:GOSUB 2010:RETURN

01250 REM *** PRINT **********************************************
01260 PRINT"PRINTER ON..";
01270 LPRINT N1$:LPRINT
01280 FOR I=1 TO R
01290   LPRINT"#";I;
01300   FOR J=1 TO 5
01310     LPRINT TAB(6);D0$(I,J)
01320   NEXT J:LPRINT
01330 NEXT I:RETURN

01990 REM *** Minor subroutines ***
02000 PRINT"TOO LONG";:PLAY 23,2;0,12
02010 CURS C:PRINT[A63 32];:CURS C:RETURN
02020 PRINT"PRESS ANY KEY TO CONTINUE";
02030 POKE 257,1:Z7$=KEY:IF Z7$=""THEN 2030 ELSE RETURN

02040 REM *** HELP ***********************************************
02050 RESTORE 2230:GOSUB 2180

02060 REM *** CONTROL ********************************************
02070 CURS 936:PRINT[I6 R];[F8.0 FRE(0)];[F8.0 FRE($)]
02080 GOSUB 2010:PRINT"COMMAND ";:GOSUB 2030:GOSUB 2010
02090 Z=ASC(Z7$):IF Z<67 OR Z>83 THEN LET Z=1 ELSE LET Z=Z-65
02100 IF R=0 AND (Z<>6 AND Z<>8) THEN GOSUB 2140:GOTO 2060
02110 ON Z GOSUB 2140,2130,780,880,1030,300,2040,530,2140,2140,
      1160,2140,2140,2140,1250,2140,390,140
02120 GOTO 2070
02130 CLS:RETURN
02140 IF R>0 THEN PRINT"?????";ELSE PRINT"NO RECORDS";
02150 PLAY 23,2;0,16:RETURN

02160 REM *** Centralise ***
02170 PRINT TAB(64-LEN(Z7$))/2;Z7$:RETURN

02180 REM *** Menu ***
02190 CLS:PRINT:READ Z7$:GOSUB 2160
02200 READ Z,T:FOR I=1 TO Z:READ Z7$
02210 PRINT TAB(T);Z7$
02220 NEXT I:RETURN
02230 DATA "B E E D A T",12,43,"C - CLS","D - DELETE","E - EDIT"
02240 DATA "F - FIND","G - GET FILE","H - HELP","I - INSERT"
02250 DATA "L - LIST","P - PRINT","R - RECORD FILE","S - SORT"
02260 DATA "RECDs  MEMFR  STRFRE"
```

COMMANDS

C - CLEAR SCREEN
Clears the screen and returns to the command mode.

D - DELETE RECORD
Deletes a record by record number (first find the record with the L or F command), moves up the remaining records and returns to the command mode.

E - EDIT RECORD
Edits a specified field of a specified record, re-sorts, then returns to the command mode leaving the edited record displayed.

F - FIND
Searches a specified field of all records for a record matching that entered. Displays the first matching record and prompts for 'Y or N'. On 'N' the search continues, on 'Y' returns to command mode leaving matching record displayed. Failure to find a match results in a 'SEARCH COMPLETE' message being displayed.

G - LOAD FILE
Loads a previously recorded data file from tape and displays the file name. At the start of loading an arrow marker is displayed, and as each record loads, a line of arrows progresses towards the first arrow. Data loading is complete when the arrows meet. Should a bad load occur this is indicated. Returns to the command mode.

H - HELP
Displays the command menu which would have been removed with any of the following commands being used: C,E,F,G,L,P,R.

I - INSERT
Inserts a record. Each record is composed of five fields, each of which is prompted by number. If the maximum field length of forty characters is exceeded, that field will have to be re-entered. If the total length of the first four fields exceeds 133 characters then the fifth field will be reduced to 14 characters. This is done so as not to exceed the Bee's print buffer. Unused fields are executed with a <RET>. After the fifth field has been entered, the new record is sorted into position with reference to the contents of its first field and if necessary subsequent fields. Then returns to the command mode.

L - LIST
Lists all records, three at a time to the screen. Then returns to the command mode.

P - PRINT
Lists all records in field order to the printer. Nothing fancy. It is intended that the user writes his own print

routine to suit his purpose. Then returns to the command mode.

R - SAVE FILE
Sends a file to tape after asking for a file name. Arrows are displayed as in the G routine so that progress is visible. Then returns to the command mode.

S - SORT
Re-sorts the whole file to a specific field with a Shell sort. Any of the five fields may be specified. Then returns to the command mode.

RECORD & MEMORY DISPLAY
On the bottom right of the screen, there are three numbers that are updated on completion of each command. From left to right they are 1) number of records in the current file, 2) memory free [FRE(Ø)], and 3) string space free [FRE($)]. From the latter two you will see how the Bee uses up memory. Watch these displays to prevent memory overflows.

FILES
If you experience problems loading files created by BEEDAT, then you are probably using an unsuitable tape-recorder. You can try changing the file save and load routines to 3ØØ baud. This will no doubt solve the problem, but is painfully slow. If in doubt, don't call us, buy a new tape-recorder.

BEEDAT APPLICATIONS

These are too numerous to mention, but here are a few suggestions:

Household Inventory: For insurance purposes this is almost a must. Have separate fields for description, cost or current value, purchase date, and room in which the item is normally found or stored. This file can then be sorted to provide an inventory by item, value, room or purchase date. In the format for the print routine, the value field (string) could be converted to numeric for justified printing, and with just a little bit of extra code, a total value could be produced.

Mailing List: BEEDAT could be altered to include a mailing label print routine (covered later in this chapter) or alternatively used only as a file builder/manager for a stand alone mailing label program. Such a stand-alone program would need to have a LOAD FILE routine similar to lines 3ØØ to 38Ø in BEEDAT.

Library: Enter fields for author and title. Use BEEDAT to produce lists sorted by author and title.

CONVERSION TO DISK

For those lucky people out there with disk drives, replace the LOAD and SAVE routines with the following, and watch those arrows zip across the screen.

```
00300 REM *** LOAD FILE ****************************************
00310 CLS:CLOSE 2:OPEN "I",2,"BEEDAT.FIL"
00320 IF EOF(2) THEN 370
00330 IN#2 ON:OUT#0:OUT#0 OFF
00340 I=1:INPUT R:POKE 61440+R,13
00345 IF EOF(2)THEN 370
00350 INPUT Z7$,D0$(I,1),D0$(I,2),D0$(I,3),D0$(I,4),D0$(I,5)
00360 I=I+1:POKE 61438+I,9:GOTO 345
00370 IN#0:OUT#0:CLS:PLAY 9:IF R=I-1 THEN RETURN
00380 R=I-1:CLS:PRINT"BAD LOAD":PLAY 9,3:RETURN

00390 REM *** SAVE FILE ****************************************
00400 CLS:OPEN "O",6,"BEEDAT.FIL"
00410 OUT#6
00420 PRINT R
00440 POKE 61440+R,13
00460 FOR I=1 TO R:PRINT[I5 I];",";
00470 PRINT D0$(I,1);",";D0$(I,2);",";D0$(I,3);",";D0$(I,4);",";
      D0$(I,5)
00480 POKE 61439+I,9
00500 NEXT I
00510 OUT#0:CLOSE 6
00520 CLS:RETURN
```

When you upgrade to a disk system, transfer your tape copy of BEEDAT to disk. Then replace lines 390 to 520 with the new SAVE FILE routine. Save this interim copy of BEEDAT. Then RUN it, and load your BEEDAT tape file with the 'G' command, and save it to disk with the 'R' command. Note that the above SAVE and LOAD routines work only on a named file i.e. BEEDAT.FIL in this instance. If you intend having more than one file, then you will have to write a small selection routine for both SAVE and LOAD. After you have transferred the tape file(s) to disk, you should alter the LOAD FILE routine and save BEEDAT, so that it will load only disk files in future.

MAILING LIST AND ADDRESS LABEL PRINTER

Here is a simple modification which will change BEEDAT into a versatile mailing list manager and label printer. Because of BEEDAT's ability to sort on specific fields, it is particularly useful in this application. You will be able to sort by surname, first name (well you never know!), town or postcode (state). The routine has a listing feature which allows listing of names in their correct order (first name - surname), ten at a time. It also has the ability to print labels from prompted input, and in any multiples at any TAB position.

You will need to use the record fields in the following manner:

Field 1	Surname
Field 2	First Name
Field 3	No/Street
Field 4	Town
Field 5	Postcode

Change the print routine to the following:

```
01250 REM *** PRINT ***********************************************
01260 RESTORE 2270:GOSUB 2180
01270 GOSUB 2010:PRINT"SELECT OPTION ";:GOSUB 2030
01280 Z=INT(VAL(Z7$)):IF Z<1 OR Z>4 THEN 1270
01290 CLS:IF Z=4 THEN RETURN
01300 ON Z GOSUB 1310,1410,1350:GOTO 1260
01310 Z=0:PRINT:FOR I=1 TO R
01320   PRINT"#";I;TAB(6);D0$(I,2);" ";D0$(I,1)
01330   Z=Z+1:IF Z=10 OR I=R:GOSUB 2010:GOSUB 2020:CLS:PRINT:Z=0
01340 NEXT I:RETURN
01350 INPUT"Surname      ";D0$(0,1)
01360 INPUT"First Name   ";D0$(0,2)
01370 INPUT"No/Street    ";D0$(0,3)
01380 INPUT"Town         ";D0$(0,4)
01390 INPUT"Post Code    ";D0$(0,5)
01400 S=0:F=0:GOTO 1430
01410 INPUT"Enter Start  Name Number ";S
01420 INPUT"Enter Finish Name Number ";F
01430 INPUT"Quantity to be Printed   ";Q
01440 INPUT"Printing TAB Position    ";T
01450 FOR J=S TO F
01460   FOR I=1 TO Q
01470     LPRINT TAB(T);D0$(J,2);" ";D0$(J,1)
01480     FOR K=3 TO 5
01490       LPRINT TAB(T);D0$(J,K)
01500     NEXT K
01510     LPRINT\\:REM ***** adjust space to next label *****
01520   NEXT I
01530 NEXT J
01540 RETURN
```

Next, add the following DATA lines:

```
02270 DATA "LABEL PRINTER OPTIONS",5,15," ","1 - LIST ALL NAMES"
02280 DATA "2 - PRINT LABEL FROM LIST OF NAMES"
02290 DATA "3 - PRINT LABEL FROM DATA TO BE ENTERED","4 - QUIT"
```

NOTES:

The spacing adjustment between labels is carried out on line 01510. The setting shown is for labels with a spacing of 1.5 inches.

If you have a disk system, then the mailing list file created by this program (BEEDAT.FIL) can be read and used by MAILMERGE to produce 'Form' letters with WORDSTAR.

CASSETTE LABEL

You may photocopy the following label to give your BEEDAT cassette box a professional appearance:

BEEDAT

A DATA MANAGER UTILITY
FOR THE MICROBEE

WITH INSERTION SORTING AND SINGLE KEY COMMANDS

C- CLS	I- INSERT
D- DELETE	L- LIST
E- EDIT	P- PRINT
F- FIND	R- RECORD FILE
G- GET FILE	S- SORT
H- HELP	

BEEDAT

MAKING CHANGE

A recurring problem encountered by many small (and some not so small) businesses, is working out how much cash (and in what denominations) is to be withdrawn from the bank to fill wage packets. It can be a time-consuming and error-prone exercise by the pencil-and-paper method. This program will alleviate those problems, providing answers both on the screen and printer. It has been set up to handle twenty inputs (20 employees), but this is easily altered to suit your requirements.

Program name : CHANGE

```
00100 REM CHANGE MAKER
00110 DIM A1(20),C(12,20),C1(12),D1(12),T(12),T1(12)
00120 CLS:P=14:PRINT "AMOUNT";
00130 CURS 1,16:PRINT "Enter 0 to exit ";
00140 PLAY 0,16:CURS 1,16:PRINT [A32 32];
00150 FOR I=1 TO 12
00160   READ C1$(I),D1(I)
00170   CURS P:PRINT C1$(I);:P=P+4
00180 NEXT I
00190 K=1:PRINT
00200 INPUT""A1(K):PRINT TAB(13);
00210 IF A1(K)=0 THEN 330
00220 GOSUB 250
00230 K=K+1:IF K>20 THEN 330
00240 GOTO 200
00250 Z6=A1(K)
00260 FOR I=1 TO 12
00270   Z=INT(Z6/D1(I)):Z6=Z6-(FLT(Z)*D1(I))
00280   PRINT [I4 Z];
00290   T(I)=T(I)+Z:T1(I)=T1(I)+(FLT(Z)*D1(I))
00300   C(K,I)=Z
00310 NEXT I
00320 PRINT:RETURN
00330 PRINT\ "Press <RETURN> for Totals";
00340 Z7$=KEY:IF Z7$ <> CHR(13) THEN 340
00350 CLS
00360 PRINT "UNIT   NUMBER      TOTAL"
00370 FOR I=1 TO 12
00380   PRINT C1$(I);TAB(8) [I4 T(I)];TAB(16) [F9.2 T1(I)]
00390   T1(0)=T1(0)+T1(I)
00400 NEXT I
00410 PRINT "=======TOTAL==$";[F9.2 T1(0)]
00420 PRINT\ "Press <RETURN> for PRINTER or <BREAK> to EXIT";
00430 Z7$=KEY:IF Z7$ <> CHR(13) THEN 430
00440 P=20:LPRINT\ "  #   AMOUNT";
00450 FOR I=1 TO 12
00460   LPRINT TAB(P) C1$(I);
00470   P=P+4
00480 NEXT I
00490 LPRINT
```

```
00500 FOR I=1 TO K-1
00510   P=18:LPRINT\ [I3 I];[F9.2 A1(I)];
00520   FOR J=1 TO 12
00530     LPRINT TAB(P) [I4 C(I,J)];
00540     P=P+4
00550   NEXT J
00560 NEXT I
00570 LPRINT\ TAB(18)[A48 61]
00580 LPRINT "UNITS";:P=18
00590 FOR I=1 TO 12
00600   LPRINT TAB(P) [I4 T(I)];
00610   P=P+4
00620 NEXT I
00630 LPRINT\\\\\
00640 LPRINT "Please supply cash in the following denominations."\
00650 LPRINT "UNIT   NUMBER    $ TOTAL"\
00660 FOR I=1 TO 12
00670   LPRINT C1$(I);TAB(8) [I4 T(I)];TAB(16) [F9.2 T1(I)]
00680 NEXT I
00690 LPRINT\ "=========TOTAL $";[F9.2 T1(0)]
00700 LPRINT\\\\\
00710 END
00720 DATA"$50",50,"$20",20,"$10",10,"$5",5,"$2",2,"$1",1
00730 DATA"50c",.50,"20c",.20,"10c",.10,"5c",.05,"2c",.02,"1c",.01
```

The following is a part of the print out:

Please supply cash in the following denominations.

UNIT	NUMBER	$ TOTAL
$50	36	1800.00
$20	2	40.00
$10	4	40.00
$5	3	15.00
$2	3	6.00
$1	1	1.00
50c	4	2.00
20c	8	1.60
10c	4	0.40
5c	8	0.40
2c	3	0.06
1c	4	0.04

=========TOTAL $ 1906.50

UTILITIES

THE STRIPPER

Not 'that' kind of stripper. This is a utility which strips the unnecessary spaces from BASIC programs. It ignores the spaces found between quotation marks and square brackets and also leaves those very necessary spaces in PLOT and NEXT* statements (see Volume One). It also minimises REM statements in three ways:

1. Where the REM shares a line with other commands, it removes the REM statement completely.

2. Where the whole line is a REM statement, a check is made to see if that line number is called from the program. If the line is called, then it is shortened to leave just the REM token.

3. If the line is not called, then the line is deleted from the program.

This kind of utility is often referred to as a 'packer' for obvious reasons. Removing spaces will save precious memory for those data-base programs, help to fit longer programs into those smaller Bees, and improve RUNning and LOADing time.

Program Name: TRIMIT

```
00090;************************ TRIMIT ****************************
00092;*                                                          *
00094;*                  by MICHAEL J DAVISON                    *
00096;************************************************************

00100           ORG     7800H           ;TRIM  (C) MJD 1984
```

Load the starting address of the program into the EXEC address pointer.

```
00110           LD      HL,00A6H        ;EXEC STORE ADDR.
00120           LD      BC,START        ;GET START OF THIS PROG
00130           LD      (HL),C          ;SAVE THIS ADDR
00140           INC     HL              ;INTO
00150           LD      (HL),B          ;EXEC ADDRESS
00160           JP      8021H           ;BACK TO BASIC
```

Start of program. BASIC lines consist of a two byte line number, followed by a one byte line length then text and a carriage return (ØDH).

```
00170 START     LD      DE,0900H        ;START OF BASIC PROG
00180 NEWL      INC     DE              ;SKIP OVER
00190           INC     DE              ;LINE No.
00200           OR      A               ;CLEAR CARRY
00210           LD      HL,(08D2H)      ;GET END PTR.
00220           SBC     HL,DE           ;CALCULATE DIFF.
00230           JP      C,804BH         ;DONE IF HL<DE
00240           PUSH    DE              ;SAVE PTR. TO LINE LEN.
00250 GET       INC     DE              ;GET NEXT CHAR.
00260           LD      A,(DE)          ;INTO REGISTER
00270           CP      20H             ;IS IT A SPACE?
00280           JR      Z,MOVE          ;REMOVE IT!
```

If a REM token is found, first check if the whole line is a REM. If it is a REM after an instruction, then remove the REM part of the line, else test to see if the line is called. If there is no call to the line then delete it.

```
00290 NOT       CP      0A1H            ;REM AT END OF LINE?
00300           JR      NZ,CONCHK       ;CONTINUE IF NOT
00310           POP     HL              ;GET LINE LEN PTR
00320           PUSH    HL              ;SAVE IT AGAIN
00330           INC     HL              ;BUMP BY ONE
00340           OR      A               ;CLEAR  CARRY
00350           SBC     HL,DE           ;COMPARE HL +DE
00360           LD      A,H             ;TEST TO SEE IF
00370           OR      L               ;RESULT ZERO
00380           JR      Z,REMCHK        ;JP -  FULL LINE IS REM

00385 ;GOES THROUGH HERE IF ONLY PART OF LINE IS A REM.
00390           POP     HL              ;GET LINE LEN PTR
00400           LD      (LENG),HL       ;SAVE LEN PTR IN HERE
00410           INC     DE              ;BUMP CUR. PTR OVER TOKEN
00420           PUSH    DE              ;SAVE CURRENT PTR
00430           DEC     DE              ;DEC POINTER FOR COLON
00440           DEC     DE              ;WITH REM STATEMENT
00450           LD      C,(HL)          ;GET LINE LEN
00460           LD      B,00            ;COMPLETE COUNT
00470           ADD     HL,BC           ;HL POINTS TO END OF LINE
00480           DEC     HL              ;-1 TO LEAVE 0D AT END
00490           PUSH    HL              ;SAVE SOURCE PTR
00500           OR      A               ;CLEAR CARRY
00510           SBC     HL,DE           ;CALC No. BYTES DESTROYED
00520           LD      B,L             ;TRANSFER BYTE COUNT
00530           LD      HL,(LENG)       ;GET BACK LINE LEN PTR
00540 DECR      DEC     (HL)            ;DEC LINE LEN
00550           DJNZ    DECR            ;FOR No.BYTES KILLED
00560           POP     HL              ;RETRIEVE SOURCE PTR
00570           JP      STACK           ;BIG SCRUNCH
```

If not a REM then test for end of line or test for NEXT* and PLOT

tokens because these have mandatory blanks which need to be left. Also test for quotation marks and square brackets and ignore spaces in these.

```
00580 CONCHK    CP      0DH             ;END OF LINE?
00590           JR      Z,ALL           ;JP IF END OF LINE
00600           CP      8BH             ;IS IT NEXT
00610           JR      Z,NEXT          ;CHECK FOR NEXT*
00620           CP      0D8H            ;IS IT PLOT?
00630           JR      NZ,COLON        ;OUT IF NOT
00640           INC     DE              ;MANDATORY SPACE
00650           JR      GET             ;LOOK FOR MORE SPACES
00660 NEXT      INC     DE              ;AFTER NEXT
00670           LD      A,(DE)          ;GET CHARACTER
00680           CP      '*'             ;IS IT *
00690           JR      NZ,GET+1        ;OUT IF NOT NEXT*
00700           INC     DE              ;BUMP POINTER
00710           CALL    8245H           ;SKIP OVER BLANKS
00720 CLICK     INC     DE              ;OVER FIRST CHAR
00730           LD      A,(DE)          ;GET CHAR
00740           CP      20H             ;LOOKING FOR SPACE
00750           JR      NZ,CLICK        ;LOOP TILL SPACE
00760           JR      GET             ;BACK IF FOUND
00770 COLON     CP      22H             ;OPENING "
00780           JR      NZ,SQBR         ;OUT IF NOT
00790 LOOP      INC     DE              ;NEXT CHARACTER
00800           LD      A,(DE)          ;INTO REG
00810           CP      22H             ;LOOKING FOR CLOSING "
00820           JR      Z,GET           ;OUT IF FOUND
00830           CP      0DH             ;END OF LINE?
00840           JR      Z,ALL           ;YOU FORGOT A "
00850           JR      LOOP            ;LOOK FOR " OR 0D
00860 SQBR      CP      '['             ;PRINT[xx yy]?
00870           JR      NZ,GET          ;OUT IF NOT OPEN BRACKET
00880 TRICK     INC     DE              ;NEXT CHAR
00890           LD      A,(DE)          ;GET CHAR
00900           CP      ']'             ;LOOK FOR CLOSE BRACKET
00910           JR      NZ,TRICK        ;LOOP BACK TILL FOUND
00920           JR      GET             ;BACK TO MAIN STREAM
00930 ALL       INC     DE              ;FINISHED LINE
00940           POP     HL              ;CLEAR STACK
00950           JR      NEWL            ;GET NEXT LINE
```

The move routine. Shifts text to fill space then decrements the line length and end of program pointer.

```
00960 MOVE      PUSH    DE              ;SAVE CURRENT PTR.
00970           LD      HL,(08D2H)      ;GET END PTR.
00980           DEC     HL              ;1 LESS CHAR.
00990           LD      (08D2H),HL      ;SAVE NEW VALUE
01000           INC     HL              ;TO END OF TEXT
01010           OR      A               ;CLEAR CARRY
01020           SBC     HL,DE           ;CALC. No. OF BYTES
01030           PUSH    HL
01040           POP     BC              ;BC=BYTES
```

```
01050           LD      H,D
01060           LD      L,E             ;HL=DESTIN
01070           INC     HL              ;HL=SOURCE
01080           LDIR                    ;FILL IN GAP
01090           POP     DE              ;GET BACK PTR.
01100           POP     HL              ;GET LINE LEN.
01110           DEC     (HL)            ;LESS 1
01120           PUSH    HL              ;SAVE IT AGAIN
01130           JP      GET+1           ;DO REST OF PGM.
```

The REM check routine. Tests if a REM line is called in the program. The line number is converted into an ASCII string. This string is then searched for. If it is found then a REM token is left on the line.

```
01140 REMCHK    POP     HL              ;GET LINE LEN PTR
01150           PUSH    HL              ;SAVE IT AGAIN
01160           PUSH    DE              ;SAVE CURRENT TEXT PTR
01170           DEC     HL              ;LINE No. LOW
01180           LD      A,(HL)          ;SAVE IT IN A
01190           DEC     HL              ;LINE No. HIGH
01200           LD      H,(HL)          ;GET LINE No. HIGH
01210           LD      L,A             ;GET LINE LOW.(HL=LINE No.
01220           EXX                     ;MORE REGS TO PLAY WITH
01230           LD      HL,STORE        ;LOAD ADDR OF ASCII STORE
01240           EXX                     ;BACK TO OTHER REGS
01250           LD      DE,2710H        ;10000's
01260           CALL    DIVIDE          ;CALC AND CONVERT
01270           LD      DE,03E8H        ;1000's
01280           CALL    DIVIDE          ;CALC AND CONVERT
01290           LD      DE,0064H        ;100's
01300           CALL    DIVIDE          ;CALC AND CONVERT
01310           LD      DE,000AH        ;TENS
01320           CALL    DIVIDE          ;CALC AND CONVERT
01330           LD      DE,0001H        ;UNITS
01340           CALL    DIVIDE          ;CALC AND CONVERT
01350           POP     DE              ;RETRIEVE CURRENT TEXT PTR
01360           JR      OVER            ;JUMP OVER DIVIDE

01365 ;DIVISION BY MULTIPLE SUBTRACTION METHOD
01370 DIVIDE    XOR     A               ;CLEAR ACCUM+CARRY
01380 AGAIN     SBC     HL,DE           ;DIVIDEND-DIVISOR
01390           JR      C,FIN           ;OUT IF NOT ENOUGH
01400           INC     A               ;COUNT+1
01410           JR      AGAIN           ;SUBTRACT TILL NOT ENOUGH
01420 FIN       ADD     HL,DE           ;RESTORE REMAINDER
01430           ADD     A,30H           ;CONVERT QUOTENT TO ASCII
01440           EXX                     ;GET STORE REG
01450           LD      (HL),A          ;SAVE DIGIT IN STORE
01460           INC     HL              ;BUMP STORE TO NEXT LOCN
01470           EXX                     ;SWAP BACK REGS
01480           RET                     ;FINISHED FOR NOW
```

When testing for a line number, checks are made that the right token is present for it to be a line number. Checks are made to

ensure that the number found is not part of a larger number.

```
01490 OVER    PUSH   DE              ;SAVE TEXT PTR
01500         LD     DE,0900H        ;GET START OF PROG
01510         INC    DE              ;OVER LINE NO.
01520 FAIL    INC    DE              ;NEXT CHAR
01530         LD     HL,STORE        ;GET LINE No. ASCII PTR
01540         LD     B,05            ;MAX OF 5 DIGITS
01550 LOOPS   LD     A,(HL)          ;GET FIRST DIGIT
01560         CP     30H             ;IS IT "0"
01570         JR     NZ,CLOCK        ;OUT IF FOUND FIRST NUMBER
01580         DEC    B               ;DEC NO. DIGITS
01590         INC    HL              ;BUMP STORE PTR
01600         JR     LOOPS           ;SKIP ALL LEADING ZEROS
01610 CLOCK   LD     A,(DE)          ;GET CHAR FROM PROG
01620         CP     (HL)            ;DOES IT MATCH STORE
01630         JR     Z,LESCHK        ;DO MORE TESTS IF IT DOES
01640 THERE   INC    DE              ;BUMP TO NEXT CHAR
01650         PUSH   HL              ;SAVE STOR PTR
01660         LD     HL,(08D2H)      ;GET END OF FILE
01670         OR     A               ;CLEAR CARRY
01680         SBC    HL,DE           ;CALC EOF-TEXT PTR
01690         POP    HL              ;RETRIEVE STORE PTR
01700         JR     C,FINCHK        ;OUT IF EOF REACHED
01710         JR     CLOCK           ;KEEP LOOKING
01720 LESCHK  DEC    DE              ;LOOK AT PREVIOUS CHAR
01730         LD     A,(DE)          ;GET CHAR
01740         CP     30H             ;CHAR MUST NOT BE
01750         JR     C,NOTNUM        ;A NUMBER OTHERWISE FAIL
01760         CP     3AH             ;TEST ADN TRY AGAIN
01770         JR     NC,NOTNUM       ;IF>9
01780 NOWAY   INC    DE              ;RESTORE TEXT PTR
01790         INC    DE              ;TO NEXT CHAR
01800         JR     CLOCK           ;KEEP LOOKING
01810 NOTNUM  INC    DE              ;RESTORE TEXT PTR
01820         PUSH   DE              ;SAVEIT
01830 BSTEP   DEC    DE              ;BACK ONE CHAR
01840         LD     A,(DE)          ;GET THAT CHAR
01850         CP     80H             ;TOKEN LIMIT
01860         JR     C,BSTEP         ;KEEP LOOKING FOR TOKEN
01870         CP     8DH             ;GOTO TOKEN?
01880         JR     Z,RTOKEN        ;RIGHT TOKEN
01890         CP     92H             ;GOSUB?
01900         JR     Z,RTOKEN        ;RIGHT TOKEN
01910         CP     89H             ;THEN?
01920         JR     Z,RTOKEN        ;RIGHT TOKEN
01930         CP     88H             ;ELSE?
01940         JR     Z,RTOKEN        ;RIGHT TOKEN
01950         CP     8BH             ;NEXT?
01960         JR     NZ,NOCHEK       ;WRONG TOKEN
01970         INC    DE              ;GO TO NEXT CHAR
01980         LD     A,(DE)          ;GET IT
01990         CP     '*'             ;LOOKING FOR NEXT*
02000         JR     NZ,NOCHEK       ;IF NOT NEXT*
02010 RTOKEN  POP    DE              ;GET TEXT PTR
```

```
02020 AGA       LD      A,(DE)          ;GET CHAR
02030           CP      (HL)            ;COMPARE WITH STORE
02040           JR      NZ,FAIL         ;OUT IF NO MATCH
02050           INC     HL              ;BUMP STORE PTR
02060           INC     DE              ;BUMP TEXT PTR
02070           DJNZ    AGA             ;TRY FOR THE WHOLE NO.
02080           LD      A,(DE)          ;GET NEXT CHAR
02090           CP      30H             ;SAME AS BEFORE THIS CHAR
02100           JR      C,YES           ;CANNOT BE A NUMBER
02110           CP      3AH             ;SO THIS CHECKS IT
02120           JR      NC,YES          ;TO YES OF NOT NUMBER
02130           JR      FAIL+1          ;FAIL TEST AGAIN
02140 NOCHEK    POP     DE              ;TO HERE IF TOKEN WRONG
02150           JR      THERE           ;KEEP TRYING

02155 ;TO HERE IF LINE No. FOUND.................
02160 YES       POP     DE              ;PASS TEST.GET CURRENT PTR
02170           POP     HL              ;GET LINE LEN PTR
02180           LD      (LENG),HL       ;SAVE LEN PTR FOR LATER
02190           PUSH    DE              ;SAVE CURRENT PTR
02200           PUSH    HL              ;SAVE LEN PTR
02210           LD      C,(HL)          ;GET NO.BYTES IN LINE
02220           LD      B,00            ;COMPLETE COUNTER
02230           ADD     HL,BC           ;SHIFT HL TO END OF LINE
02240           DEC     HL              ;LEAVE 0D AT END
02250           POP     DE              ;GET PTR OFF STACK
02260           POP     DE              ;GET CURRENT PTR
02270           INC     DE              ;LEAVE REM TOKEN
02280           PUSH    DE              ;SAVE PTR
02290           PUSH    HL              ;SAVE SOURCE PTR
02300           LD      HL,(LENG)       ;GET LINE LEN PTR
02310           LD      (HL),03         ;MAKE LEN 3BYTES
02320           POP     HL              ;RESTORE SOURCE PTR
02330           JP      STACK           ;THE BIG SCRUNCH

02335 ;TO HERE IF LINE No. NOT FOUND.........
02340 FINCHK    POP     DE              ;GET CURRENT PTR
02350           POP     HL              ;GET LINE LEN PTR
02360           PUSH    HL              ;SAVE IT AGAIN
02370           LD      C,(HL)          ;GET LINE LEN
02380           LD      B,00            ;COMPLETE COUNT
02390           ADD     HL,BC           ;SOURCE PTR TO END OF LINE
02400           POP     DE              ;LINE LEN PTR
02410           DEC     DE              ;BACK OVER LINE NO.
02420           DEC     DE              ;SO
02430           DEC     DE              ;SET UP TEXT PTR FOR EXIT
02440           PUSH    DE              ;SAVE THIS PTR
02450           INC     DE              ;BACK TO DESTIN PTR

02455 ;THE STACK ROUTING DELETES A LARGE BLOCK OF TEXT
02460 STACK     PUSH    HL              ;SAVE SOURCE PTR
02470           PUSH    DE              ;SAVE DESTIN PTR
02480           EXX                     ;SWAP REGS
02490           POP     DE              ;GET DESTIN
02500           POP     HL              ;GET SOURCE
```

```
02510           OR      A               ;CLEAR CARRY
02520           SBC     HL,DE           ;CALC NO.BYTES KILLED
02530           EX      DE,HL           ;INTO DE'
02540           EXX                     ;SWAP BACK REGS
02550           PUSH    HL              ;SAVE SOURCE
02560           PUSH    DE              ;SAVE DESTIN
02570           LD      DE,(08D2H)      ;GET END PTR
02580           EX      DE,HL           ;SWAP VALUES
02590           OR      A               ;CLEAR CARRY
02600           SBC     HL,DE           ;CALC NO. BYTES TO SHIFT
02610           INC     HL              ;PLUS ONE
02620           PUSH    HL              ;TRANSFER TO
02630           POP     BC              ;B COUNT
02640           POP     DE              ;GET BACK DESTIN
02650           POP     HL              ;GET SOURCE
02660           LDIR                    ;SCRUNCH IT UP
02670           EXX                     ;GET BYTES LOST
02680           LD      HL,(08D2H)      ;GET END OF FILE
02690           OR      A               ;CLEAR CARRY
02700           SBC     HL,DE           ;CALC NEW END OF FILE
02710           LD      (08D2H),HL      ;SAVE IT
02720           POP     DE              ;RESTORE TEXT PTR
02730           INC     DE              ;BUMP TO NEXT LOCN
02740           JP      NEWL            ;GO DO THE NEXT LINE
02750 STORE     DEFS    05              ;PLACE TO STORE ASCII LINE
02760 LENG      DEFS    04              ;TEMPORARY STORAGE AREA
02770           END                     ;COMPLETED
```

OPERATION

1 - LOAD assembled program
2 - EXEC
3 - LOAD BASIC program to be stripped
4 - EXEC

In a matter of seconds the prompt '>' will be seen. If you now LIST the program you will find the job is complete.

BASIC LOADER

For those people without EDASM, or for that matter those who want to take a short cut, we present here a BASIC loader. After saving the program, RUN it, then LOAD the program to be trimmed and EXECute.

Program Name: TRIMBL

```
00090 REM********************* TRIMBL ****************************
00092 REM                                                        *
00094 REM              by MICHAEL J DAVISON                      *
00096 REM*********************************************************
00100 FOR I=1 TO 417
00110   READ D
00120   T1=T1+FLT(D)
00130   POKE(30720+I-1),D
00140 NEXT I
00150 IF T1<>46823 THEN 170
00160 U=USR(30720)
00170 PRINT "CHECKSUM ERROR"
00180 END
10000 DATA 33,166,0,1,12,120,113,35,112,195
10010 DATA 33,128,17,0,9,19,19,183,42,210
10020 DATA 8,237,82,218,75,128,213,19,26,254
10030 DATA 32,40,107,254,161,32,38,225,229,35
10040 DATA 183,237,82,124,181,40,119,225,34,166
10050 DATA 121,19,213,27,27,78,6,0,9,43
10060 DATA 229,183,237,82,69,42,166,121,53,16
10070 DATA 253,225,195,119,121,254,13,40,57,254
10080 DATA 139,40,7,254,216,32,21,19,24,193
10090 DATA 19,26,254,42,32,188,19,205,69,130
10100 DATA 19,26,254,32,32,250,24,175,254,34
10110 DATA 32,12,19,26,254,34,40,165,254,13
10120 DATA 40,14,24,244,254,91,32,155,19,26
10130 DATA 254,93,32,250,24,147,19,225,24,131
10140 DATA 213,42,210,8,43,34,210,8,35,183
10150 DATA 237,82,229,193,98,107,35,237,176,209
10160 DATA 225,53,229,195,28,120,225,229,213,43
10170 DATA 126,43,102,111,217,33,161,121,217,17
10180 DATA 16,39,205,212,120,17,232,3,205,212
10190 DATA 120,17,100,0,205,212,120,17,10,0
10200 DATA 205,212,120,17,1,0,205,212,120,209
10210 DATA 24,16,175,237,82,56,3,60,24,249
10220 DATA 25,198,48,217,119,35,217,201,213,17
10230 DATA 0,9,19,19,33,161,121,6,5,126
10240 DATA 254,48,32,4,5,35,24,247,26,190
10250 DATA 40,13,19,229,42,210,8,183,237,82
10260 DATA 225,56,99,24,239,27,26,254,48,56
10270 DATA 8,254,58,48,4,19,19,24,225,19
10280 DATA 213,27,26,254,128,56,250,254,141,40
10290 DATA 22,254,146,40,18,254,137,40,14,254
10300 DATA 136,40,10,254,139,32,26,19,26,254
10310 DATA 42,32,20,209,26,190,32,171,35,19
10320 DATA 16,248,26,254,48,56,9,254,58,48
10330 DATA 5,24,157,209,24,172,209,225,34,166
10340 DATA 121,213,229,78,6,0,9,43,209,209
10350 DATA 19,213,229,42,166,121,54,3,225,195
10360 DATA 119,121,209,225,229,78,6,0,9,209
10370 DATA 27,27,27,213,19,229,213,217,209,225
10380 DATA 183,237,82,235,217,229,213,237,91,210
```

```
10390 DATA 8,235,183,237,82,35,229,193,209,225
10400 DATA 237,176,217,42,210,8,183,237,82,34
10410 DATA 210,8,209,19,195,15,120
```

DECIMAL/HEXADECIMAL CONVERSION

The following program differs from its namesake in WILDCARDS Volume One, as this one is a free-standing program that allows you to convert a Hexadecimal number to Decimal or vice versa: the one in Volume One merely printed the conversion table at Appendix Eleven. In case you ever have to convert several numbers, you will find it quicker to use this program than to use the conversion table in Volume One. (There is a set of Hex-Dec/Dec-Hex etc conversion routines in the subroutine library found in Volumes One and Two, but you would need to write a short program to call them.)

Program name : HEXDEC

```
00100 REM Program by Doug Stanborough
00110 CLS:CURS 10,3:PRINT"HEX TO DECIMAL OR DECIMAL TO HEX"
00120 PRINT\"I can convert HEX from 0 to FFFF to decimal"
00130 PRINT"I can convert  DECIMAL from 0 to 65535 to HEX"
00140 CURS 5,8:PRINT"Which do you want ?"
00150 CURS 10,10:PRINT"1.     Convert HEX to DECimal ?"
00160 CURS 10,11:PRINT"2.     Convert DECimal to HEX ?"
00170 PRINT\"Make your selection: ( 1 or 2 )"
00180 X1$=KEY:IF X1$=""THEN 180
00190 IF X1$="1"THEN 220
00200 IF X1$="2"THEN 620
00210 GOTO 170
00220 CLS
00230 INPUT"Enter HEX No to be changed to DECIMAL:"A1$
00240 IF LEN(A1$)>4THEN PRINT"Number too large (FFFF is the limit
      )":GOTO220
00250 IF ASC(A1$)>70THEN PRINT"That is NOT a HEX number":GOTO 230
00260 IF ASC(A1$)<40THEN PRINT"That is NOT a HEX number":GOTO 230
00270 B1$=A1$(;LEN(A1$),LEN(A1$))
00280 GOSUB 540
00290 J=I
00300 B1$=A1$(;LEN(A1$)-1,LEN(A1$)-1)
00310 GOSUB 540
00320 K=I
00330 B1$=A1$(;LEN(A1$)-2,LEN(A1$)-2)
00340 GOSUB 540
00350 L=I
00360 B1$=A1$(;1,1)
00370 GOSUB 540
00380 M=I
00390 P1=FLT(M)*16*16*16
```

```
00400 Q1=FLT(L)*16*16
00410 R1=FLT(K)*16
00420 S1=FLT(J)
00430 IF LEN(A1$)=4 : T1=P1+Q1+R1+S1:GOTO 470
00440 IF LEN(A1$)=3 : T1=Q1+R1+S1:GOTO 470
00450 IF LEN(A1$)=2 : T1=R1+S1:GOTO 470
00460 IF LEN(A1$)=1 : T1=S1
00470 PRINT"The number is "T1
00480 PRINT\"Press the SPACE bar for another number, press any
      other key for"
00490 PRINT"DECIMAL to HEX "
00500 X1$=KEY:IF X1$=""THEN 500
00510 IF X1$=" "THEN 220
00520 GOTO 120
00530 GOTO 110
00540 IF B1$="F" : I=15:RETURN
00550 IF B1$="E" : I=14:RETURN
00560 IF B1$="D" : I=13:RETURN
00570 IF B1$="C" : I=12:RETURN
00580 IF B1$="B" : I=11:RETURN
00590 IF B1$="A" : I=10:RETURN
00600 I=INT(VAL(B1$)):RETURN
00610 END
00620 CLS
00630 INPUT"Enter DECIMAL No to be changed to HEX:"Y1
00640 A1$=" "
00650 IF Y1>65535 THEN PRINT"That number is higher than allowed
      (65535 is the limit)": PLAY 0,10: GOTO 620
00660 IF Y1<16    : Y=INT(Y1):GOTO 820
00670 IF Y1<=255  : B=INT(Y1):GOTO 780
00680 IF Y1<=4095 : C=INT(Y1):GOTO 750
00690 Y=INT(Y1/16/16/16)
00700 GOSUB 890
00710 K1=FLT(Y)*16*16
00720 J1=K1*16
00730 C1=Y1-J1
00740 C=INT(C1)
00750 Y=C/16/16
00760 GOSUB 890
00770 A=Y*16*16:B=C-A
00780 Y=B/16
00790 GOSUB 890
00800 A=Y*16:C=B-A
00810 Y=C
00820 GOSUB 890
00830 PRINT\
00840 PRINT\"The HEX No is "A1$
00850 PRINT\"Press  SPACE bar for another DEC to HEX, otherwise
      any  other key"
00860 X1$=KEY:IF X1$=""THEN 860
00870 IF X1$=" "THEN 620
00880 GOTO 110
00890 IF Y=15 : B1$="F":GOTO 970
00900 IF Y=14 : B1$="E":GOTO 970
00910 IF Y=13 : B1$="D":GOTO 970
```

```
00920 IF Y=12 : B1$="C":GOTO 970
00930 IF Y=11 : B1$="B":GOTO 970
00940 IF Y=10 : B1$="A":GOTO 970
00950 B1$=STR(Y)
00960 IF LEN(B1$)=2 : B1$=B1$(;2,2)
00970 A1$=A1$+B1$
00980 RETURN
```

A MACHINE CODE DISASSEMBLER IN BASIC

by Michael J Davison

This program may be rather long, but is very effective. A disassembler translates a machine language program back into the original source code, with some limitations. It will not normally print assembler directives such as ORG, DEFR, END etc, and will not show the original names of labels. Some disassemblers can generate source code that can be saved to tape, and reloaded under EDASM, but you will have to verify the code to be safe. If you already have a machine code version and do not understand how it works, this BASIC version may help you. It works by using an algorithm based on the instruction description set out in that excellent book "Programming the Z80" by Rodnay Zaks.

When the program is RUN, you will be prompted for Start and Finish addresses in hexadecimal, and given the option of having the output on screen or on a printout. Try disassembling BASIC, which starts at 8000H and ends at BFFFH. This will take about four hours on most printers, so maybe you ought to start on something smaller! The program to be disassembled is to be loaded first, but if it lives in low memory around 0900-1000 Hex, it will be overwritten by the disassembler, and must be Moved to high memory using the Monitor.

Program Name : DISASS

```
00010 REM********************* DISASS ***************************
00020 REM*                                                      *
00030 REM*              by Michael J Davison                    *
00040 REM*                                                      *
00040 REM********************************************************
00100 DIM Z5(3)
00110 P1$=""
00120 STRS(600):GOSUB 2400
00130 FOR I=0 TO 7:READ E1$(I):NEXT I
00140 DATA "ADD  A,","ADC  A,","SUB  ","SBC  A,","AND  "
00150 DATA "XOR  ","OR   ","CP   "
00160 FOR I=0 TO 7:READ C0$(I):NEXT I
00170 DATA "RLC","RRC","RL ","RR ","SLA","SRA","?","SRL"
00180 C1$(0)="BIT":C1$(1)="RES":C1$(2)="SET"
00190 FOR I=0 TO 7:READ R1$(I):NEXT I
```

```
00200 DATA "B","C","D","E","H","L","(HL)","A"
00210 FOR I=0 TO 3:READ D1$(I),D2$(I),D3$(I),D4$(I):NEXT I
00220 DATA "BC","BC","BC","BC","DE","DE","DE","DE"
00230 DATA "HL","IX","IY","HL","SP","SP","SP","AF"
00240 FOR I=0 TO 3:READ G0$(I):NEXT I:FOR I=0 TO 3:READ G1$(I):
      NEXT I
00250 DATA "LD","CP","IN","OUT","I","D","IR","DR"
00260 FOR I=0 TO 3:READ T2$(I):NEXT I
00270 DATA "DAA","CPL","SCF","CCF"
00280 FOR I=0 TO 5:READ J2$(I):NEXT I
00290 DATA "DJNZ ","JR   ","JR   NZ","JR   Z","JR   NC","JR   C"
00300 FOR I=0 TO 7:READ C7$(I):NEXT I
00310 DATA "NZ","Z","NC","C","PO","PE","P","M"
00320 GOSUB[A0]2010:M1$=Z7$:GOSUB 1920:ON G+1 GOTO 430,390,420,330
00330 IF O<>203 THEN 740
00340 GOSUB 1920:I1$=R1$(R)
00350 IF G>0 THEN 380
00360 IF F=6 THEN 1710
00370 I2$=C0$(F)+"    "+I1$:GOTO 1720
00380 I2$=C1$(G-1)+" "+STR$(F)+","+I1$:GOTO 1720
00390 IF O<>118 THEN 410
00400 I2$="HALT":GOTO 1720
00410 I2$="LD   "+R1$(F)+","+R1$(R):GOTO 1720
00420 I2$=E1$(F)+R1$(R):GOTO 1720
00430 IF R<>6 THEN 460
00440 I2$="LD   "+R1$(F)+","
00450 GOSUB 1920:I2$=I2$+Z7$:GOTO 1720
00460 IF R<>1 THEN 510
00470 IF F4<>0 THEN 500
00480 I2$="LD   "+D1$(INT(FLT(F)/2))+","
00490 GOSUB 1920:X1$=Z7$:GOSUB 1920:I2$=I2$+Z7$+X1$:GOTO 1720
00500 I2$="ADD  HL,"+D1$(F/2):GOTO 1720
00510 IF R<>3 THEN 540
00520 I2$="INC":IF F4<>0 THEN LET I2$="DEC"
00530 I2$=I2$+"  "+D1$(F/2):GOTO 1720
00540 IF R<>4 AND R<>5 THEN 570
00550 I2$="INC":IF R=5 THEN LET I2$="DEC"
00560 I2$=I2$+"  "+R1$(F):GOTO 1720
00570 IF R<>2 THEN 650
00580 IF O<>42 AND O<>34 THEN 630
00590 X1$="HL"
00600 F2=F4:GOSUB 1920:Y1$=Z7$:GOSUB 1920:Y1$="("+Z7$+Y1$+")":
      F4=F2
00610 IF F4=0 THEN LET Z7$=Y1$:Y1$=X1$:X1$=Z7$
00620 I2$="LD   "+X1$+","+Y1$:GOTO 1720
00630 X1$="A":IF F>=6 THEN 600
00640 Y1$="("+D1$(F/2)+")":GOTO 610
00650 IF R<>7 THEN 700
00660 IF F>3 THEN 690
00670 IF F<2 THEN LET I2$=C0$(F)+"A":GOTO 1720
00680 T4$=C0$(F):I2$=T4$(;1,2)+"A":GOTO 1720
00690 I2$=T2$(F-4):GOTO 1720
00700 IF O=0 THEN LET I2$="NOP":GOTO 1720
00710 IF O=8 THEN LET I2$="EX   AF,AF'":GOTO 1720
00720 I2$=J2$(F-2):IF F>3 THEN LET I2$=I2$+","
```

```
00730 GOSUB 1920:I2$=I2$+Z7$+"    ":I2$=I2$(;1,11):GOSUB 1980:
      GOTO 1720
00740 IF O=221 THEN 1150
00750 IF O=237 THEN 1310
00760 IF O=253 THEN1700
00770 IF O=195 OR O=233 OR O=205 OR O=201 THEN 960
00780 IF O<>254 THEN 800
00790 GOSUB 1920:I2$="CP   "+Z7$:GOTO 1720
00800 IF R <>1 AND R<>5 THEN 870
00810 IF F4<>0 THEN 850
00820 IF R=1 THEN LET X1$="POP"
00830 IF R=5 THEN LET X1$="PUSH"
00840 I2$=X1$+" "+D4$(INT(FLT(F/2))):GOTO 1720
00850 IF O=217 THEN LET I2$="EXX":GOTO 1720
00860 IF O=249 THEN LET I2$="LD   SP,HL":GOTO 1720
00870 IF R<>6 THEN 890
00880 F2=FLT(F):GOSUB 1920:F=INT(F2):I2$=E1$(F)+Z7$:GOTO 1720
00890 IF R<>2 THEN 920
00900 I2$="JP   "
00910 I2$=I2$+C7$(F)+",":GOTO 490
00920 IF R<>0 THEN 940
00930 I2$="RET  "+C7$(F):GOTO 1720
00940 IF R<>4 THEN 1030
00950 I2$="CALL ":GOTO 910
00960 IF O<>195 THEN 980
00970 I2$="JP   ":GOTO 490
00980 IF O<>205 THEN 1000
00990 I2$="CALL ":GOTO 490
01000 IF O=233 THEN LET I2$="JP   (HL)"
01010 IF O=201 THEN LET I2$="RET"
01020 GOTO 1720
01030 IF R<>7 THEN 1050
01040 B2=FLT(F)*8:GOSUB [B2]2010:I2$="RST  "+Z7$:GOTO 1720
01050 IF O<>211 THEN 1080
01060 GOSUB 1920:I2$="OUT  ("
01070 I2$=I2$+Z7$+"),A":GOTO 1720
01080 IF O<>219 THEN 1090
01090 IF O<>219 THEN 1110
01100 GOSUB 1920:I2$="IN   A,":I2$=I2$+"("+Z7$+")":GOTO 1720
01110 IF O=235 THEN LET I2$="EX   DE,HL"
01120 IF O=243 THEN LET I2$="DI"
01130 IF O=251 THEN LET I2$="EI"
01140 GOTO 1720
01150 X1$="X"
01160 GOSUB 1920
01170 IF G<>1 THEN 1250
01180 IF R=6  THEN 1210
01190 GOSUB 1920:I2$="LD   (I"+X$+"+"+Z7$+"),"+R1$(R)
01200 GOTO 1720
01210 IF F=6 THEN 1710
01220 I2$="LD   "+R1$(F)+",(I"
01230 I2$=I2$+X1$+"+":GOSUB 1920
01240 I2$=I2$+Z7$+")":GOTO 1720
01250 IF O<>54 THEN 1280
01260 I2$="LD   (I"+X1$+"+"
```

```
01270 GOSUB 1920:I2$=I2$+Z7$+"),":GOSUB 1920:I2$=I2$+Z7$:GOTO 1720
01280 IF G<>2 THEN 2110
01290 IF R<>6 THEN 2110
01300 E=F:GOSUB 1920:I2$=E1$(E)+"(I"+X1$+"+"+Z7$+")":GOTO 1720
01310 GOSUB 1920:IF G<>1 THEN 1650
01320 IF R<>7 THEN 1420
01330 IF F>3  THEN 1390
01340 I2$="LD   ":ON F+1 GOTO 1350,1360,1370,1380
01350 I2$=I2$+"I,A":GOTO 1720
01360 I2$=I2$+"R,A":GOTO 1720
01370 I2$=I2$+"A,I":GOTO 1720
01380 I2$=I2$+"A,R":GOTO 1720
01390 ON F-3 GOTO 1400,1410,1710,1710
01400 I2$="RRD":GOTO 1720
01410 I2$="RLD":GOTO 1720
01420 IF R<>3 THEN 1450
01430 IF F=4 OR F=5 THEN 1710
01440 X1$=D1$(F/2):GOTO 600
01450 IF R<>4  THEN 1480
01460 IF O<>68 THEN 1710
01470 I2$="NEG":GOTO 1720
01480 IF R<>2 THEN 1520
01490 I2$="ADC":IF F4<>0 THEN 1510
01500 I2$="SBC"
01510 I2$=I2$+"  HL,"+D1$(F/2):GOTO 1720
01520 IF R<>5 THEN 1570
01530 I2$="":IF O=69 THEN LET I2$=RETN"
01540 IF O=77 THEN LET I2$="RETI"
01550 IF I2$=""THEN 1710
01560 GOTO 1720
01570 IF R<>1 THEN 1590
01580 I2$="OUT  (C),"+R1$(F):GOTO 1720
01590 IF R<>0 THEN 1610
01600 I2$="IN   "+R1$(F)+",(C)":GOTO 1720
01610 I2$="":IF F=0 THEN I2$="IM0"
01620 IF F=2 THEN I2$="IM1"
01630 IF F=3 THEN LET I2$="IM2"
01640 GOTO 1550
01650 IF G<>2 THEN 1710
01660 IF R>3  THEN 1710
01670 IF F<4  THEN 1710
01680 IF R<3 OR F<6 THEN LET I2$=G0$(R)+G1$(F-4):GOTO 1720
01690 I2$="OT"+G1$(F-4):GOTO 1720
01700 X1$="Y":GOTO 1160
01710 I2$="??? "+STR$(O)
01720 PRINT M1$,,I2$,,P1$
01730 IF L3=1 THEN LPRINT M1$,,I2$,,P1$
01740 P1$=""
01750 IF A0<=A3 THEN 320
01760 PRINT "DONE"
01770 IF L3=1 THEN GOSUB 2090
01780 END
01790 G=INT(B1/64):T3=B1-FLT(G)*64:F=INT(T3/8):R=INT(T3)-F*8
01800 F4=FLT(F-(F/2)*2):RETURN
01810 Z=LEN(A1$):IF Z<>4 THEN 1910
```

```
01820 Z0=0:Z1$="0123456789ABCDEF"
01830 Z5(0)=1:Z5(1)=16:Z5(2)=256:Z5(3)=4096
01840 FOR I=1 TO 4
01850 J=SEARCH(Z1$,A1$(;I,I))-1
01860 IF J=-1 THEN NEXT*I 1910
01870 Z0=Z0+(Z5(4-I)*FLT(J))
01880 NEXT I
01890 IF Z0<0 THEN LET Z0=65536+Z0
01900 RETURN
01910 Z0=-1:RETURN
01920 Z=INT(A0)
01930 O=PEEK(Z):T1=FLT(O):GOSUB[T1] 2010:B1=T1:GOSUB 1790
01940 REM***
01950 M1$=M1$+" "+Z7$:A0=A0+1:IF O<32 OR O>127: P1$=P1$+"?"
01960 IF O>31 AND O<128 THEN LET P1$=P1$+CHR(O)
01970 RETURN
01980 D6=FLT(O)+A0:IF O>127 THEN LET D6=FLT(O)+A0-256
01990 B2=FLT(INT(D6/256)):GOSUB[B2]2010:I2$=I2$+"   ("+Z7$
02000 B2=D6-B2*256:GOSUB[B2] 2010:I2$=I2$+Z7$+")":RETURN
02010 VAR(Z0)
02020 Z7$=""
02030 IF Z0>255 THEN LET L=4 ELSE LET L=2
02040 FOR I=1 TO L
02050 Z0=Z0/16:X=INT(FRACT(Z0)*16)+1
02060 Z0=FLT(INT(Z0))
02070 Z7$=Z1$(;X,X)+Z7$
02080 NEXT I:RETURN
02090 LPRINT "DONE"
02100 RETURN
02110 IF G<>3   THEN 2210
02120 IF O<>203 THEN 2140
02130 GOSUB 1920:I1$="(I"+X1$+"+"+Z7$+")":GOSUB 1920:GOTO 350
02140 I2$="":IF O=255 THEN LET I2$="POP  I"+X1$
02150 IF O=227 THEN LET I2$="EX   (SP),I";X1$
02160 IF O=229 THEN LET I2$="PUSH I"+X1$
02170 IF O=233 THEN LET I2$="JP   (I"+X1$+")"
02180 IF O=249 THEN LET I2$="LD   SP,I"+X1$
02190 IF I2$=""THEN 1710
02200 GOTO 1720
02210 IF R<>1 THEN 2260
02220 IF F4=0 THEN 2250
02230 Y1$=D2$((F-1)/2):IF X1$="Y"THEN Y1$=D3$((F-1)/2)
02240 I2$="ADD  I"+X1$+","+Y1$:GOTO 1720
02250 I2$="LD   I"+X1$+",":GOTO 490
02260 IF R=0  THEN 1710
02270 IF R<>2 THEN 2330
02280 IF O<>34 AND O<>42 THEN 1710
02290 P=O:GOSUB 1920:Y1$=Z7$:GOSUB 1920
02300 IF P=34 THEN LET I2$="LD   ("+Z7$+Y1$+"),I"+X1$
02310 IF P=42 THEN LET I2$="LD   I"+X1$+",("+Z7$+Y1$+")"
02320 GOTO 1720
02330 IF R<>3 THEN 2370
02340 I2$="":IF O=35 THEN LET I2$="INC  I"
02350 IF O=43 THEN LET I2$="DEC  I"+X1$
02360 GOTO 1550
```

```
02370 IF O<>52 AND O<>53 THEN I2$="DE"
02380 I2$="IN":IF O=53 THEN LETI2$="DE"
02390 I2$=I2$+"C  (I":GOTO 1230
02400 CLS:PRINT"Z80 DISASSEMBLER"
02410 INPUT"START ADDRESS"A1$
02420 GOSUB1810:A2=Z0:IF Z0<0 THEN 2410
02430 INPUT"STOP  ADDRESS"A1$
02440 GOSUB1810:A3=Z0:IF Z0<0 THEN 2430
02450 IF A3<A2 THEN 2410
02460 A0=A2
02470 INPUT"PRINTER (Y/N)"A1$
02480 IF A1$="Y"OR A1$="y"THEN LET L3=1 ELSE LET L3=0
02490 DIM R1(7),C0(7),C1(2),E1(7),D1(3),T2(3),J2(5),D4(3),C7(7),
      G0(3),G1(3),D3(3),D2(3)
02500 RETURN
```

JULIAN DATE SUBROUTINE

In WILDCARDS Volume One, we started a Subroutine Library which was increased in Volume Two. In this volume we are only repeating the Date routine as it is relevant to the Julian Date routines.

****

The Date entry and validation routine. Validates day 1-31, month 1-12, year 81-99. Returns the following:

Z6$	-	Date	e.g 17/03/83
D	-	Day	e.g. 17
M	-	Month	e.g. 3
Y	-	Year	e.g. 83

```
64800 REM *** Date ***
64810 CLS:CURS 465:PRINT "ENTER DATE: DD/MM/YY":CURS 476
64820 PRINT" ";:Z6$="":FOR I=1 TO 3:FOR J=1 TO 2
64830 Z7$=KEY:IF Z7$<"0" OR Z7$>"9" THEN 64830
64840 PRINT Z7$;:Z6$=Z6$+Z7$:NEXT J
64850 IF I<3 THEN LET Z6$=Z6$+"/":PRINT"/";
64860 NEXT I
64870 D=INT(VAL(Z6$(;1,2))):M=INT(VAL(Z6$(;4,5))):
      Y=INT(VAL(Z6$(;7,8)))
64880 IF D>0 AND D<32 AND M>0 AND M<13 AND Y>80 AND D<99:RETURN
64890 CURS 960:PRINT"INVALID DATE";:PLAY23;0,16:GOTO 64800
```

****

The Julian Date Calendar is used in some programs to find such things as on which day of the week a given date occurs. The following routines can be used to perform such functions. They can also be used in business applications to weed out overdue accounts. The first routine converts a date of the format DD/MM/YY into Julian Date. The second reverses the process, and the third provides the day of the week.

Variables passed are D, M and Y as per Date routine. The Julian Date is returned in J1.

```
64900 REM *** Date to Julian Date ****
64910 D1=FLT(D):M1=FLT(M):Y1=FLT(Y+1900)
64920 J1=30.57*M1:J1=J1-FRACT(J1)
64930 J2=365.25*Y1-395.25:J2=J2-FRACT(J2)
64940 J1=J1+J2+D1
64950 IF M>2 THEN LET J1=J1-2 ELSE GOTO 64970
64960 Y2=Y1/4:IF Y2=Y2-FRACT(Y2) THEN LET J1=J1+1
64970 RETURN
```

****

The Julian Date is passed to the variable J1 and the Date is returned in Z6$ as well as D,M and Y. Remember to DIMension the Z1$ array.

```
64980 REM *** Julian Date to Date ***
64990 Y1=J1/365.26+1:Y1=Y1-FRACT(Y1)
65000 D1=395.25-365.25*Y1:D1=D1+J1-FRACT(D1)
65010 Y2=Y1/4:IF Y2=Y2-FRACT(Y2) THEN LET D2=1 ELSE LET D2=2
65020 IF D1>(91-D2) THEN LET D1=D1+D2
65030 M1=D1/30.57:M1=M1-FRACT(M1)
65040 D2=30.57*M1:D2=D2-FRACT(D2):D1=D1-D2
65050 IF M1>12:M1=1:Y1=Y1+1
65060 D=INT(D1):M=INT(M1):Y=INT(Y1-1900)
65070 Z1$(0)=STR(D):Z1$(1)=STR(M):Z1$(2)=STR(Y):Z6$=""
65080 FOR I=0 TO 2:Z2$=Z1$(I):Z=LEN(Z2$):Z1$(I)=Z2$(;2,Z)
65090 IF Z<3:Z1$(I)="0"+Z1$(I)
65100 Z6$=Z6$+Z1$(I):IF I<2:Z6$=Z6$+"/"
65110 NEXT I:RETURN
```

****

Returns the day of the week in Z5$ from the Julian Date passed in J1

```
65120 REM *** Day of the Week ***
65130 X1=J1/7:X1=(X1-FRACT(X1))*7
65140 X=INT(J1-X1):X=X*3+1
65150 Z7$="SATSUNMONTUEWEDTHUFRI"
65160 Z5$=Z7$(;X,X+2)
65170 RETURN
```

NOTES FOR DATE PROGRAMS

The 'Julian Date' is the number of days counted from 1 January 0001 A.D., but you should know that it ignores one very important fact. In 1582 Pope Gregory XIII decreed that 15th October should be the day after 4th October of that year. The year 1582

therefore only had 355 days instead of 365.

Every fourth year is a leap year, except century years that are not divisible by 400.

SORTING AND SELECTING BY DATE
by Adrian Van Wierst

Dates that occur within the same century (e.g. between 1900 and 1999) can conveniently be sorted, or tested for selection, if they are stored in the YYMMDD format (i.e. 2 digits for year, month and day, in that order). For example, 7 March 1984 would be stored as 840307. To illustrate the simplicity of dealing with dates in this format, the following routine selects and prints dates between 7 March 1984 and 26 June 1985. The routine assumes that each of the dates to be tested is stored as a string variable in an array X0(n). The dates are converted to real numbers for the purposes of a numerical comparison.

```
00100 FOR A=0 TO N
00110 IF VAL(X0$(A))=>840307 AND VAL(X0$(A))<=850626:PRINT X0$(A)
00120 NEXT A
```

It may be necessary to convert dates to or from the YYMMDD format: for example, if you have already stored dates in another format, or if you wish to allow dates to be input or to be printed out in another format. The following routines can be used to convert between the YYMMDD format and either of the following formats:

D/M/YY	No leading zeros for month and day; always two digits for year
DD/MM/YY	Leading zeros for month or day less than 10; always two digits for year

Where either routine is used for input/printing conversion, it would ordinarily be called immediately before printing, as the case requires. Conversion between other date formats can be done by adapting the programming techniques contained in the routines.

Program name: DATCON

```
00100 REM date conversion routines
00110 REM by Adrian Van Wierst
00120 GOSUB 500:REM format selection
00130 ON A GOSUB 160,230,310,410
00140 PRINT X0$,D1$,M1$,Y1$,X1$
00150 GOTO 120

00160 REM *** DD/MM/YY to YYMMDD ***
```

```
00170 INPUT "Enter date in original format";X0$
00180 D1$=X0$(;1,2)
00190 M1$=X0$(;4,5)
00200 Y1$=X0$(;7,8)
00210 X1$=Y1$+M1$+D1$
00220 RETURN

00230 REM *** YYMMDD to DD/MM/YY ***
00240 INPUT "Enter date in original format";X0$
00250 S1$="/"
00260 D1$=X0$(;5,6)
00270 M1$=X0$(;3,4)
00280 Y1$=X0$(;1,2)
00290 X1$=D1$+S1$+M1$+S1$+Y1$
00300 RETURN

00310 REM *** D/M/YY to YYMMDD ***
00320 INPUT "Enter date in original format";X0$
00330 S0$="0"
00340 IF LEN(X0$)=6:D1$=S0$+X0$(;1,1):M1$=S0$+X0$(;3,3)
00350 IF SEARCH (X0$,"/")=2 AND SEARCH (X0$,"/",2)=5:D1$=S0$+X0$
      (;1,1):M1$=X0$(;3,4)
00360 IF SEARCH (X0$,"/")=3 AND SEARCH (X0$,"/",2)=5:D1$=X0$
      (;1,2):M1$=S0$+X0$(;4,4)
00370 IF LEN(X0$)=8:D1$=X0$(;1,2):M1$=X0$(;4,5)
00380 Y1$=X0$(;LEN(X0$)-1,LEN(X0$))
00390 X1$=Y1$+M1$+D1$
00400 RETURN

00410 REM *** YYMMDD to D/M/YY ***
00420 INPUT "Enter date in original format";X0$
00430 S1$="/"
00440 IF X0$(;5,5)="0" : D1$=X0$(;6,6) ELSE LET D1$=X0$(;5,6)
00450 IF X0$(;3,3)="0" : M1$=X0$(;4,4) ELSE LET M1$=X0$(;3,4)
00460 IF X0$(;1,1)="0" : Y1$=X0$(;2,2) ELSE LET Y1$=X0$(;1,2)
00470 X1$=D1$+S1$+M1$+S1$+Y1$
00480 RETURN

00490 REM *** menu and input ***
00500 PRINT "1 DD/MM/YY to YYMMDD"\"2 YYMMDD to DD/MM/YY"
      \"3 D/M/Y to YYMMDD"\"4 YYMMDD to D/M/Y"
00510 INPUT "Enter number of selection";A
00520 RETURN


00080 REM routine to print dates between selected dates YYMMDD
00090 GOSUB 220
00100 FOR A=0 TO N
00200 IF VAL(X0$(A))=>840307 AND VAL(X0$(A))<=850626:PRINT X0$(A)
00210 NEXT A
00215 END
00220 DIM X0(10):N=0
00230 INPUT X0$(N)
00240 IF X0$(N)="END" THEN RETURN
00250 N=N+1:GOTO 230
```

NUMERIC KEYPAD

The following program redefines a portion of your MicroBee keyboard as a numeric keypad. It will be of great use to copy DATA statements as found in the previous program, not only because the numeric keys are redefined, but because pressing the LINE FEED key causes the keyword DATA to appear on the screen. Naturally, the routine can be enabled or disabled at will. In the keypad mode, you can still have access to the lower case alpha keys by pressing the SHIFT key. The program is well-commented and is supplemented by a BASIC loader for the benefit of those who want to use it immediately and for those who cannot use EDASM.

Program name : KEYPDS

```
00100 ;*******************************************************
00110 ;   ********    MICROBEE NUMERIC KEY PAD    *******
00120 ;                      Mark Bishop 5/84
00130 ;*******************************************************
00140 ;This program relocates itself  to top of memory,  and on
00150 ;selection  changes the function  of some  keyboard keys,
00160 ;to define a NUMERIC KEYPAD and a DATA key.  This enables
00170 ;faster  & easier entry of BASIC numeric data statements.
00180 ;The speaker produces a 'click' to verify each key entry.
00190 ;           CTRL & K   -  KEYPAD   (numeric key pad on )
00200 ;           CTRL & N   -  NORMAL   (numeric key pad off)
00210 ;           LINE FEED  -  adds DATA statement to program
00220 ;
00230 ;When numeric key pad is on, these keys are changed -
00240 ;           M  key outputs numeric value of 0
00250 ;           K   ..    ..      ..      ..   .. 1
00260 ;           L   ..    ..      ..      ..   .. 2
00270 ;           ;   ..    ..      ..      ..   .. 3
00280 ;           O   ..    ..      ..      ..   .. 4
00290 ;           P   ..    ..      ..      ..   .. 5
00300 ;           [   ..    ..      ..      ..   .. 6
00310 ;           0   ..    ..      ..      ..   .. 7
00320 ;           :   ..    ..      ..      ..   .. 8
00330 ;           -   ..    ..      ..      ..   .. 9
00340 ;If the  SHIFT key  is held during  key-pad mode,  alpha
00350 ;keys output their normal character in lower case.
00360 ;Add stick-on labels to keys,  with the revised function
00370 ;marked in red to aid recognition, or mark keys directly.
00380 ;Cold start your 'Bee, then  LOAD  and  RUN this program
00390 ;ONLY ONCE,   prior to starting  your  BASIC programming.
00400 ;'KEYPAD' will remain  until a cold start. (ESC & RESET)
00410          ORG     10000D          ;(2710H) - GOOD AS ANY
00420 TOPMEM   EQU     00A0H           ;TOP OF MEMORY POINTER
00430 KEYVEC   EQU     00C2H           ;KEYBOARD INPUT VECTOR
00440 CASE     EQU     0101H           ;KEYBOARD SHIFT LOCK
00450 WRMBAS   EQU     8021H           ;ROM - WARM BASIC START
00460 KEYSCN   EQU     0A3E9H          ;ROM - KEYBOARD SCAN
00470 ;   Initialize program,  by getting  JP vector  for the
```

```
00480 ;   particular 'Bee model and storing it.  Alter top of
00490 ;   memory pointer to protect program.  Move program to
00500 ;   protected area, and alter JP vector to entry point.
00510 ;   JP to BASIC with a warm start.
00520         LD      HL,(KEYVEC)     ;GET KEYBOARD JP VECTOR
00530         LD      (KEY+1),HL      ;STORE
00540         LD      HL,(TOPMEM)
00550         LD      DE,200H
00560         OR      A               ;RESET CARRY FLAG
00570         SBC     HL,DE           ;REDUCE TOPMEM PTR 200H
00580         LD      (TOPMEM),HL     ;& STORE
00590         PUSH    HL              ;SAVE NEW TOPMEM PTR
00600         EX      DE,HL           ;ALSO IN DE
00610         LD      HL,FLAG         ;PROGRAM START
00620         LD      BC,LENMKR-FLAG  ;PROGRAM LENGTH
00630         LDIR                    ;MOVE PROGRAM
00640         POP     HL              ;NEW PGM LOCATION
00650         INC     HL              ;PROGRAM ENTRY POINT
00660         LD      (KEYVEC),HL     ;CHANGE KEYBOARD VECTOR
00670         JP      WRMBAS          ;WARM BASIC START
00680 ;*****************************************************
00690 ;** This part of the program is shifted to high memory **
00700 FLAG    DEFB    00H             ;BITS 7-3 DATA & SPC
00710                                 ;BIT  1   KEYPAD ON
00720                                 ;BIT  0   CASE STORE
00730 START   PUSH    HL              ;SAVE CALLING REGISTER
00740         LD      HL,(KEYVEC)     ;PROGRAM START ADDRESS
00750         DEC     HL              ;HL NOW POINTS TO FLAG
00760         LD      A,(HL)          ;GET CURRENT FLAG
00770         AND     248             ;MASK OFF BITS 0-2
00780         OR      A               ;BITS 7-3 ZERO ?
00790         JR      NZ,DATA         ;NO,DATA STMNT TO CMPLETE
00800         BIT     1,(HL)          ;KEYPAD MODE SET ?
00810         JR      Z,KEY           ;NO
00820         LD      A,1             ;YES, THEN
00830         LD      (CASE),A        ;FORCE UPPER CASE
00840 KEY     CALL    KEYSCN
00850         JR      NZ,BACK2        ;NO KEY PRESSED, SO RET
00860 CKEYPD  CP      11              ;^K SET KEYPAD MODE ?
00870         JR      NZ,CKNORM       ;NO
00880         SET     1,(HL)          ;YES, SET KEYPAD FLAG
00890         LD      A,(CASE)        ;GET CURRENT CASE STATUS
00900         AND     1               ;MASK OFF BITS 7-1
00910         OR      (HL)            ;MERGE WITH REST OF FLAGS
00920         LD      (HL),A          ;STORE FLAGS
00930         JR      BACK2
00940 CKNORM  CP      14              ;^N SET NORMAL MODE ?
00950         JR      NZ,KYPDRQ
00960         RES     1,(HL)          ;YES, RESET KEYPAD FLAG
00970         LD      A,(HL)          ;FLAGS
00980         LD      (CASE),A        ;REPLACE CASE STATUS
00990         OR      1               ;RESET Z FLG,NO KEY PRESS
01000         JR      BACK2
01010 KYPDRQ  BIT     1,(HL)          ;KEYPAD MODE SET ?
01020         JR      Z,BACK2         ;NO, SO RET
```

```
Ø1Ø3Ø CKDATA   CP      1Ø              ;LF, DATA STATEMENT REQ ?
Ø1Ø4Ø          JR      NZ,CK7          ;NO,CK IF KEY CHNGE REQ'D
Ø1Ø5Ø          LD      A,248           ;DATA STATEMENT FLAG
Ø1Ø6Ø          OR      (HL)            ;MERGE WITH OLD FLAG
Ø1Ø7Ø          LD      (HL),A          ;SAVE NEW FLAG
Ø1Ø8Ø          JR      BACK2
Ø1Ø9Ø DATA     BIT     7,(HL)          ;D SENT ?
Ø11ØØ          JR      Z,DATAA1        ;YES
Ø111Ø          LD      A,'D'           ;NO, SEND IT
Ø112Ø          RES     7,(HL)          ;RESET FLAG
Ø113Ø          JR      BACK1
Ø114Ø DATAA1   BIT     6,(HL)          ;A SENT ?
Ø115Ø          JR      Z,DATAT         ;YES
Ø116Ø          LD      A,'A'
Ø117Ø          RES     6,(HL)
Ø118Ø          JR      BACK1
Ø119Ø DATAT    BIT     5,(HL)          ;T SENT ?
Ø12ØØ          JR      Z,DATAA2        ;YES
Ø121Ø          LD      A,'T'
Ø122Ø          RES     5,(HL)
Ø123Ø          JR      BACK1
Ø124Ø DATAA2   BIT     4,(HL)          ;2ND A SENT ?
Ø125Ø          JR      Z,SPACE         ;YES
Ø126Ø          LD      A,'A'
Ø127Ø          RES     4,(HL)
Ø128Ø          JR      BACK1
Ø129Ø SPACE    LD      A,' '
Ø13ØØ          RES     3,(HL)
Ø131Ø          JR      BACK1
Ø132Ø BACK1    LD      L,A             ;SAVE KEY PRESSED
Ø133Ø          LD      A,64            ;SPEAKER ON BIT 6, PORT 2
Ø134Ø          OUT     (2),A           ;OUTPUT 'CLICK' TO SPEAKR
Ø135Ø          LD      B,ØØH           ;MAXIMUM COUNT
Ø136Ø          DJNZ    $               ;FOR TIME DELAY
Ø137Ø          XOR     A               ;ZERO A, AND SET Z FLAG
Ø138Ø          OUT     (2),A           ;TO STOP OUTPUT
Ø139Ø          LD      A,L             ;KEY PRESSED BACK INTO A
Ø14ØØ                                  ;ZERO FLAG SET INDICATES
Ø141Ø                                  ;TO KEYSCN ROUTINE THAT
Ø142Ø                                  ;A KEY HAS BEEN PRESSED
Ø143Ø BACK2    POP     HL              ;RESTORE CALLING REG.
Ø144Ø          RET                     ;TO BASIC ROM
Ø145Ø CK7      CP      'Ø'             ;CK IF KEYPAD KEYS AND
Ø146Ø          JR      NZ,CK8          ;CHANGE TO KEYPAD VALS
Ø147Ø          LD      A,'7'
Ø148Ø          JR      BACK1
Ø149Ø CK8      CP      ':'
Ø15ØØ          JR      NZ,CK9
Ø151Ø          LD      A,'8'
Ø152Ø          JR      BACK1
Ø153Ø CK9      CP      '-'
Ø154Ø          JR      NZ,CK4
Ø155Ø          LD      A,'9'
Ø156Ø          JR      BACK1
Ø157Ø CK4      CP      'O'
```

```
Ø1580          JR       NZ,CK5
Ø1590          LD       A,'4'
Ø1600          JR       BACK1
Ø1610 CK5      CP       'P'
Ø1620          JR       NZ,CK6
Ø1630          LD       A,'5'
Ø1640          JR       BACK1
Ø1650 CK6      CP       '['
Ø1660          JR       NZ,CK1
Ø1670          LD       A,'6'
Ø1680          JR       BACK1
Ø1690 CK1      CP       'K'
Ø1700          JR       NZ,CK2
Ø1710          LD       A,'1'
Ø1720          JR       BACK1
Ø1730 CK2      CP       'L'
Ø1740          JR       NZ,CK3
Ø1750          LD       A,'2'
Ø1760          JR       BACK1
Ø1770 CK3      CP       ';'
Ø1780          JR       NZ,CKØ
Ø1790          LD       A,'3'
Ø1800          JR       BACK1
Ø1810 CKØ      CP       'M'
Ø1820          JR       NZ,BACK1
Ø1830          LD       A,'Ø'
Ø1840          JR       BACK1
Ø1850 LENMKR   DEFS     ØØH              ;END PGM MKR FOR LEN CALC
Ø1860          END
```

BASIC LOADER FOR NUMERIC KEYPAD

Program name : KEYPDB

```
ØØ1ØØ REM ******************************************************
ØØ11Ø REM
ØØ12Ø REM    ********    MICROBEE NUMERIC KEY PAD    *******
ØØ13Ø REM
ØØ14Ø REM                      Mark Bishop 5/84
ØØ15Ø REM
ØØ16Ø REM ******************************************************
ØØ17Ø REM
ØØ18Ø REM This program relocates itself to top of memory, and
ØØ19Ø REM on selection  changes the function of some keyboard
ØØ2ØØ REM keys, to define  a numeric keypad  and  a DATA key.
ØØ21Ø REM This enables faster & easier entry of BASIC numeric
ØØ22Ø REM data statements.  The speaker produces a 'click' to
ØØ23Ø REM verify each key entry.
ØØ24Ø REM
ØØ25Ø REM          CTRL & K   -  KEYPAD (numeric key pad on )
ØØ26Ø REM          CTRL & N   -  NORMAL (numeric key pad off)
```

```
00270 REM          LINE FEED  -  adds DATA statement to prog.
00280 REM
00290 REM When numeric key pad is enabled, these keys are changed-
00300 REM
00310 REM          M  key outputs numeric value of 0
00320 REM          K   ..    ..      ..     ..  .. 1
00330 REM          L   ..    ..      ..     ..  .. 2
00340 REM          ;   ..    ..      ..     ..  .. 3
00350 REM          O   ..    ..      ..     ..  .. 4
00360 REM          P   ..    ..      ..     ..  .. 5
00370 REM          [   ..    ..      ..     ..  .. 6
00380 REM          0   ..    ..      ..     ..  .. 7
00390 REM          :   ..    ..      ..     ..  .. 8
00400 REM          -   ..    ..      ..     ..  .. 9
00410 REM If the SHIFT key is pressed during keypad mode, alpha
00420 REM keys  output their normal  character  in  lower case.
00430 REM
00440 REM Add  stick-on  labels  to  keys,  with  the revised
00450 REM function marked in red  to aid recognition, or mark
00460 REM keys directly. Cold start your 'Bee then LOAD & RUN
00470 REM this  program  ONCE ONLY,  prior  to starting  your
00480 REM BASIC programming.  'KEYPAD'  will remain  until a
00490 REM cold start.  (ESC & RESET)
00500 REM
00510 CLS: PRINT "POKING DATA INTO MEMORY"
00520 FOR X=10000 TO 10245: READ D: POKE X,D: T=T+D: NEXT X
00530 READ C: IF C<>T THEN PRINT "DATA ENTRY ERROR": END
00540 I=USR(10000)
00550 END
00560 DATA  42,194,  0, 34, 74, 39, 42,160,  0, 17
00570 DATA   0,  2,183,237, 82, 34,160,  0,229,235
00580 DATA  33, 52, 39,  1,210,  0,237,176,225, 35
00590 DATA  34,194,  0,195, 33,128,  0,229, 42,194
00600 DATA   0, 43,126,230,248,183, 32, 57,203, 78
00610 DATA  40,  5, 62,  1, 50,  1,  1,205,233,163
00620 DATA  32,102,254, 11, 32, 11,203,206, 58,  1
00630 DATA   1,230,  1,182,119, 24, 87,254, 14, 32
00640 DATA  10,203,142,126, 50,  1,  1,246,  1, 24
00650 DATA  73,203, 78, 40, 69,254, 10, 32, 67, 62
00660 DATA 248,182,119, 24, 59,203,126, 40,  6, 62
00670 DATA  68,203,190, 24, 36,203,118, 40,  6, 62
00680 DATA  65,203,182, 24, 26,203,110, 40,  6, 62
00690 DATA  84,203,174, 24, 16,203,102, 40,  6, 62
00700 DATA  65,203,166, 24,  6, 62, 32,203,158, 24
00710 DATA   0,111, 62, 64,211,  2,  6,  0, 16,254
00720 DATA 175,211,  2,125,225,201,254, 48, 32,  4
00730 DATA  62, 55, 24,233,254, 58, 32,  4, 62, 56
00740 DATA  24,225,254, 45, 32,  4, 62, 57, 24,217
00750 DATA 254, 79, 32,  4, 62, 52, 24,209,254, 80
00760 DATA  32,  4, 62, 53, 24,201,254, 91, 32,  4
00770 DATA  62, 54, 24,193,254, 75, 32,  4, 62, 49
00780 DATA  24,185,254, 76, 32,  4, 62, 50, 24,177
00790 DATA 254, 59, 32,  4, 62, 51, 24,169,254, 77
00800 DATA  32,165, 62, 48, 24,161
00810 DATA 22894
```

GRAPHICS

INTRODUCTION

The PLOT command in MicroWorld BASIC is a very convenient and powerful facility for drawing graphics on the screen. It may be used for demonstration of the graphical representation of algebraic functions - and hence have a real educational value, or it may be used simply to create displays which are pleasing to the eye. Frequently, both these characteristics will be present in the same program - hence partially explaining the attraction of the personal computer.

The MicroBee has the distinct advantage of the capacity to PLOT in HIgh RESolution Graphics (HIRES). Hence fine drawings and graphs can be generated, simulating both art work and geometry.

We will first introduce a mildly 'artistic' program, which should capture your interest and then move on to rather larger and more detailed programs.

SPIRAL GRAPHICS

This little program will allow you to 'draw' complex designs on the screen:

Program name : SPIRAX

```
00100 REM ----- Plotting Graphics -------------------------
00110 REM       Keep Distance and Increment to small values
00112 REM       Try angles 59,60,61,89,90,91,118,120,123
00120 CLS
00130 PRINT "Input values :"
00140 INPUT "Distance, Angle, Increment";D1,A1,I1
00150 P1=3.14159
00160 W1=P1/180
00170 X=255 : Y=127 : A2=A1 :REM Initial Coordinates and Angle
00180 HIRES
00190 J=X+INT(D1*COS(W1*A2)):REM Next X axis coordinate
00200 K=Y+INT(D1*SIN(W1*A2)):REM Next Y axis coordinate
00210 IF J<0 OR J>511 OR K<0 OR K>255 THEN 280:REM Off end
00220 ON ERROR GOTO 280 :REM Prevent display loss if all PCG used
00230 PLOT X,Y TO J,K:REM Draw
00240 D1=D1+I1:  REM Reset Distance
00250 A2=A2+A1:  REM Reset Angle
00260 X=J : Y=K: REM Reset coordinates
```

```
00270 GOTO 190:  REM Repeat
00280 A0$=KEY:IF A0$="" THEN 280
00290 RUN
```

When you RUN the program, try the angles suggested in the LISTing (Line 112). Note that the PLOT command is entered only once, in Line 230 and called repeatedly within the loop from Lines 190 to 270. The REMarks in the LISTing should enable you to follow the logic of the program.

If this has whet your appetite, move on to the next program, which demonstrates conventional plotting, using scaled X and Y axes and also shows how spiral graphics of the above type can be developed from them.

COORDINATE PLOTTING

Most people these days are familiar with the manner in which standard graphs are displayed, to demonstrate growth or changes of some kind. For example, we have all seen graphs of such things as population growth, where the years are represented on the horizontal or X axis and the population figures on the vertical or Y axis. This is the most common technique, termed the Cartesian coordinate system. However, an alternative and less familiar method is to specify for each plotted point the distance from the origin and the angle that the line forms with the horizontal axis. This is the Polar coordinate system. The following program uses both:

Program name : COORDS

```
00100 REM ----- Coordinate PLOT Graphics ----------------------
00110 REM       To create Cartesian and Polar Plots
00120 REM       and Spiral Graphics
00130 REM       Try these functions :
00140 REM       #*# : #*#*# : #*#*#*# : #*#*3+#*6+14
00150 REM       SIN(#) : #*#*(4-#) : (#*#-4)/(#-2)
00160 REM       .25-SIN(#/2) : #*100-#*#*#*7 : COS(SIN(#*2)*7)
00170 REM       #*100-#*#*#*7 : COS(#*4)+(20/(#*#+3))
00180 REM       COS(#)+COS(#*3)
00190 REM       #/3.14159 (Use for Spiral Graphics)

00200 REM ----- Define the Function --------------------------
00210 FN0=#/3.14159

00220 REM ----- Select Option --------------------------------
00230 CLS
00240 E1=255 :REM Origin for X axis
00250 F1=127 :REM Origin for Y axis
```

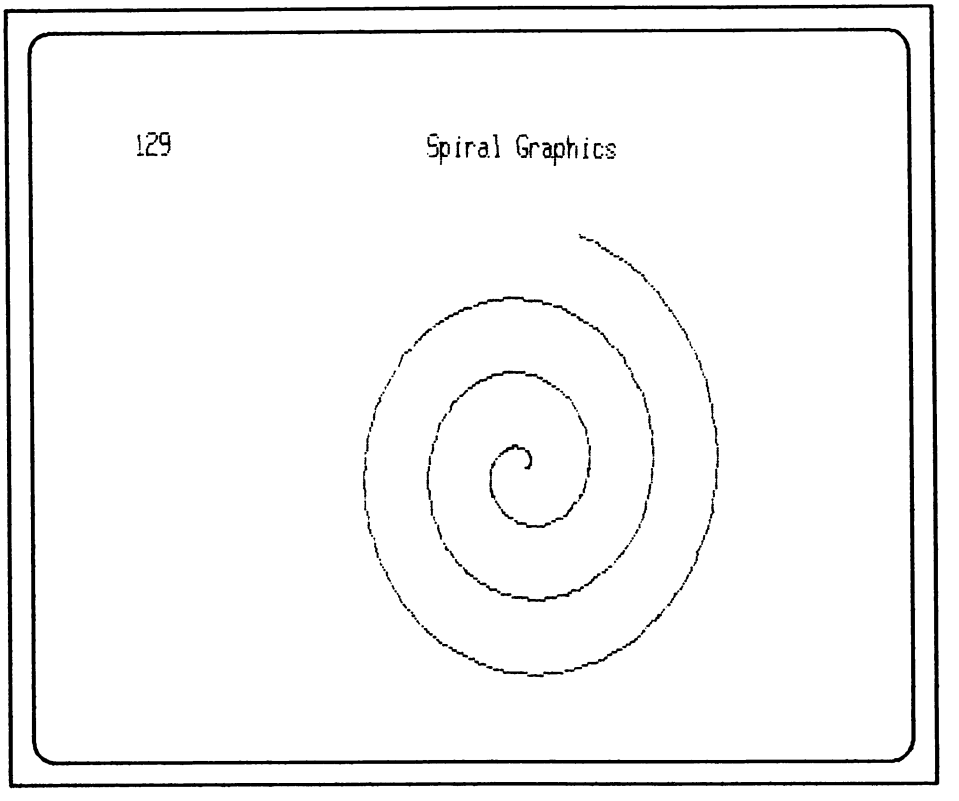
```
00260 CURS 212:PRINT "Plotting Option List"
00270 CURS 338:PRINT "1 : Cartesian Coordinates"
00280 CURS 402:PRINT "2 : Polar Coordinates"
00290 CURS 466:PRINT "3 : Spiral Graphics"
00300 INPUT "Choice :";C

00310 REM ----- Set up X, Y Axes and Scales -----------------
00320 CLS
00330 CURS 200:INPUT "Input Increment";I1
00340 PRINT\ "To expand or contract the axes, enter scale factor"
00350 INPUT "X - axis scale factor (1=Normal) :";S1
00360 INPUT "Y - axis scale factor (1=Normal) :";S2
00370 PRINT\ "To move origin left or right"
00380 INPUT "enter number from -250 to 250 (0=Central) :";C1
00390 PRINT\ "To move origin up or down"
00400 INPUT "enter number from 120 to -120 (0=Central) :";D1
00410 E1=255+C1 : E=INT(E1) :REM X displacement
00420 F1=127+D1 : F=INT(F1) :REM Y displacement
00430 CLS:HIRES
00440 IF C=3 THEN 620

00450 REM ----- Plot axes and scales -------------------------
00460 PLOT 0,F TO 511,F : REM X axis
00470 PLOT E,0 TO E,255 : REM Y axis
00480 FOR X1=E1 TO 510 STEP S1*20 :REM Scale X axis (right)
00490   PLOT INT(X1),F-1 TO INT(X1),F+1
00500 NEXT X1
00510 FOR X2=E1 TO 0 STEP S1*-20 :REM Scale X axis (left)
00520   PLOT INT(X2),F-1 TO INT(X2),F+1
00530 NEXT X2
00540 FOR Y1=F1 TO 255 STEP S2*14 :REM Scale Y axis (top)
00550   PLOT E-1,INT(Y1) TO E+1,INT(Y1)
00560 NEXT Y1
00570 FOR Y2=F1 TO 0 STEP S2*-14 :REM Scale Y axis (bottom)
00580   PLOT E-1,INT(Y2) TO E+1,INT(Y2)
00590 NEXT Y2
00600 IF C=1 THEN 790

00610 REM ----- Polar Plotting and Spiral Graphics -----------
00620 IF C=3:D1=20000 ELSE LET D1=360
00630 FOR G1=0 TO D1 STEP I1
00640   T1=G1/57.29578
00650   X1=FN0(T1)*COS(T1)
00660   Y1=FN0(T1)*SIN(T1)
00670   X2=X1*(S1*20)+E1:X=INT(X2)
00680   Y2=Y1*(S2*14)+F1:Y=INT(Y2)
00690   ON ERROR GOTO 750
00700   CURS 0:PRINT USED;" ";
00710   IF X<0 OR X>511 OR Y<0 OR Y>255 THEN 740
00720   PLOT E,F TO X,Y
00730   E=X : F=Y
00740 NEXT G1
00750 PLAY 23
00760 CURS 0:PRINT USED
00770 GOTO 770
```

```
00780 REM ----- Cartesian Plot --------------------------------
00790 FOR G1=0 TO 510 STEP I1
00800   X1=(G1-E1)/(S1*20)
00810   Y1=FN0(X1)
00820   Y2=F1+(Y1*S2*14) : Y=INT(Y2)
00830   IF Y<0 OR Y>255 THEN 890
00840   ON ERROR GOTO 900
00850   IF J=0 AND K=0 THEN 880
00860   PLOT J,K TO INT(G1),Y
00870   CURS 0:PRINT USED;" ";
00880   J=INT(G1):K=Y
00890 NEXT G1
00900 PLAY 23
00910 CURS 0:PRINT USED
00920 GOTO 920
```



You should find this well worth the effort to type in, as you can regard it as a 'utility' for standard graphics. You will find it to be very versatile. All the conventional algebraic functions used for educational purposes can be 'plotted' with ease! All that needs to be done is to enter the function in Line 210. Thus to plot the graph for the equation y = x^3 (or the function f(x)=x^3), you would enter on Line 210:

```
00210 FN0=#*#*#
```

Note that to ensure that you have the correct scale proportions

for the X and Y axes, you should check the displayed scales as they appear on your own Monitor (VDU). If necessary, you can correct any disproportion by changing the value of the multiplier (currently 14) in Lines 540, 570, 680 and 820. It is worth getting this right, so that when you plot a circle, you do not finish up with an oval!

The program contains lines to display a count of the number of PCG characters USED in the top left hand corner of the screen.

PCG GRAPHICS DATA FOR CARD GAMES

The following program will provide you with the PCG graphics details for merging with your favourite card game. The immediate mode command LISTL (list lowercase) found in versions of BASIC later than 5.22 was used to highlight keywords from the rest of the tightly-packed listing.

THE PACK OF CARDS
by Doug Stanborough

I had been trying to impress her: I had shown her my brilliance at Chess, I had shown her how to shoot down aliens by the thousand. I had even demonstrated my skill in hunting the wily Wumpus but was she impressed? No, not a bit. "But can it play Gin Rummy?" was all she asked. I found out later that that was the only game she knew, but bells rang and ideas flashed.

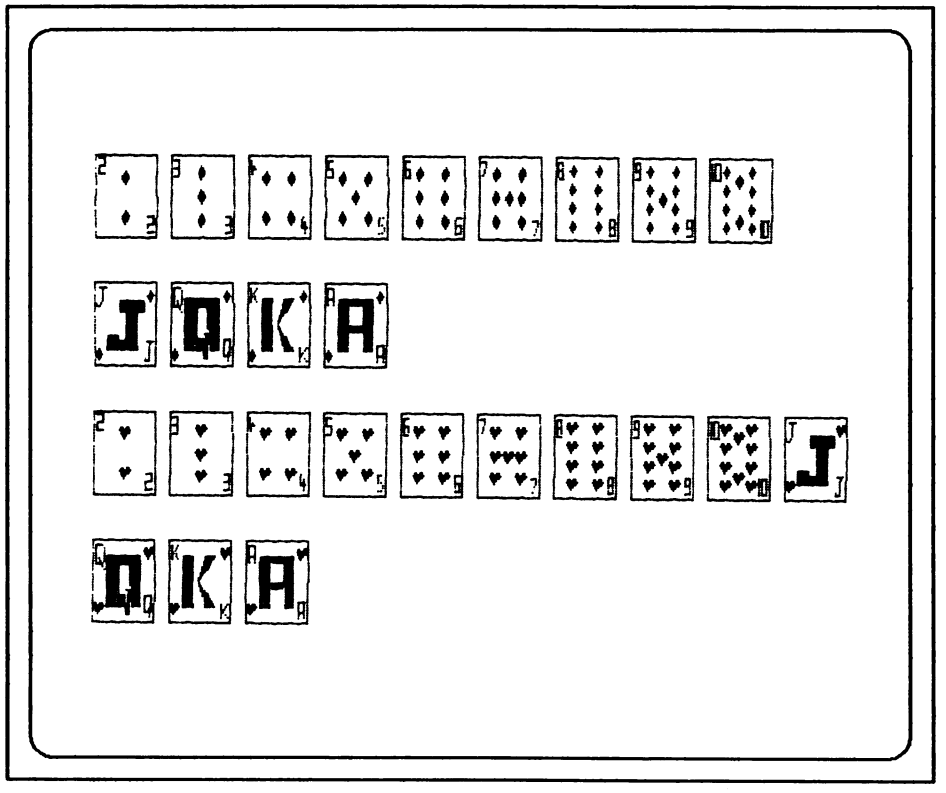
As soon as she had gone (several hours later), surrounded with graph paper I started. At first the task of producing 52 cards in graphics seemed formidable, if not impossible, for if each card used 10 characters then the lot would need 520 characters. At first I thought that six characters per card would be the ideal (3 across by 2 high), but perusal of the actual cards soon showed that ten (5 across by 2 high) characters per card was far more economical in characters. Then there were the Court cards... now my artistic ability is pretty poor, indeed the less appreciative members of my family say it is about equal to that of a two year old chimpanzee. So they were simply given large letters (J,Q,K & A). How many characters could be made common to all suits? How many could be made common to each suit?

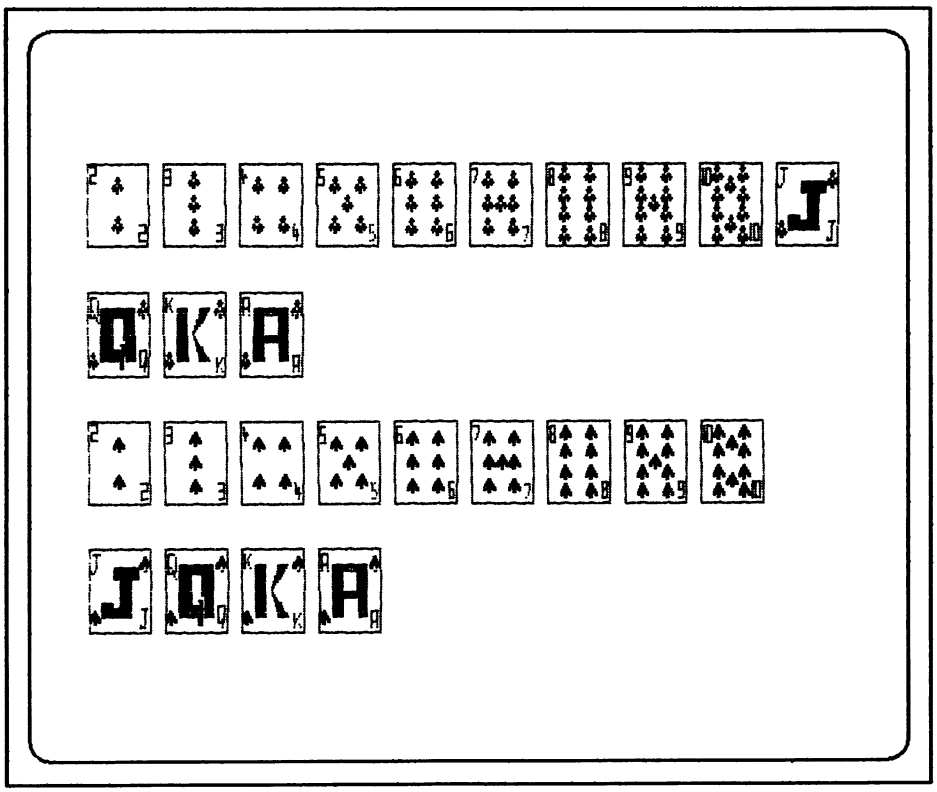
In the end I found that there were 31 characters that could be made common to all, plus 16 to make the court cards, and each suit was reduced to 10 characters, making a total of 87 characters to produce 52 cards. All of which shows just what a little planning can do.

The next step was to switch the computer on and to determine where to allocate the graphics in the PCG memory. Looking at the ASCII character table (See Appendix 10 in Wildcards Volume One), one can

see at once the advantage of keeping below Chr(126) so that the characters can be typed as such to facilitate them being given string values. Why give them string values? Of course, they could be printed straight from the graphics characters, BUT in a game, or any other program for that matter, it is so much easier to compare a string with another string, or several other strings, than it is to compare groups of individual characters.

So, to fit in the 87 characters without using Chr(34) for obvious reasons, the program was started by storing the graphics from Chr(35) to Chr(121) in the PCG memory. There is a problem character there among them (for those not entirely familiar with their keyboards yet). That is Chr(95) which does not appear on the keyboard as such, but is the DEL key pressed simultaneously with the SHIFT key, or to put it in another way, it is the DEL key in the UPPER REGISTER. This is more commonly seen on the screen and in print as the underline character '_', which should not be confused with the hyphen '-' in the following program. So now, having a pack of cards and having raised a vista of possibilities, off you go, all of you Bridge Buffs, Five Hundred Freaks, and perhaps even you Patience Players, and write your favourite game while I settle down to write Gin Rummy for my better half. Here is the program for a pack of cards:





Program name : CARDFACE

```
00100 rem by Doug Stanborough
00110 dim D0(12),D1(12),H0(12),H1(12),C0(12),C1(12),S0(12),S1(12)
00120 strs(2000)
00130 cls:curs 7,3:inverse:for I=1 to 50:print" ";:next I
00140 curs 7,12   :inverse:for I=1 to 50:print" ";:next I
00150 Y=3:for I=3 to 12:curs 7,Y:print"   ":curs 57,Y:
      print"   ":Y=Y+1:next I
00160 normal
00170 curs 25,5:print"A  PACK  OF  CARDS"
00180 curs 32,7:print"by"
00190 curs 25,9:print"Doug Stanborough"

00200 rem ALLOCATE DIAMONDS TO STRINGS
00210 D0$(0)="#=S=?":D1$(0)="@>W>$":rem ...DEUCE
00220 D0$(1)="%=T=?":D1$(1)="@>X>&":rem ...THREE
00230 D0$(2)="'S=S?":D1$(2)="@W>W(":rem ...FOUR
00240 D0$(3)=")SRS?":D1$(3)="@WVW*":rem ...FIVE
00250 D0$(4)="+T=T?":D1$(4)="@X>X,":rem ...SIX
00260 D0$(5)="-TRT?":D1$(5)="@XVX.":rem ...SEVEN
00270 D0$(6)="/U=U?":D1$(6)="@Y>Y0":rem ...EIGHT
00280 D0$(7)="1URU?":D1$(7)="@YVY2":rem ...NINE
00290 D0$(8)="3USU?":D1$(8)="@YWY4":rem ...TEN
00300 D0$(9)="5=ABQ":D1$(9)="PCDE6":rem ...JACK
00310 D0$(10)="7FGHQ":D1$(10)="PIJK8":rem .QUEEN
00320 D0$(11)="9FLMQ":D1$(11)="PINO:":rem .KING
```

```
00330 D0$(12)=";FxHQ":D1$(12)="PIyK<":rem .ACE

00340 rem ALLOCATE HEARTS TO STRINGS
00350 H0$(0)="#=]=?":H1$(0)="@>a>$":rem ...DEUCE
00360 H0$(1)="%=^=?":H1$(1)="@>b>&":rem ...THREE
00370 H0$(2)="']=]?":H1$(2)="@a>a(":rem ...FOUR
00380 H0$(3)=")]\]?":H1$(3)="@a`a*":rem ...FIVE
00390 H0$(4)="+^=^?":H1$(4)="@b>b,":rem ...SIX
00400 H0$(5)="-^\^?":H1$(5)="@b`b.":rem ...SEVEN
00410 H0$(6)="/_=_?":H1$(6)="@c>c0":rem ...EIGHT
00420 H0$(7)="1_\_?":H1$(7)="@c`c2":rem ...NINE
00430 H0$(8)="3_]_?":H1$(8)="@cac4":rem ...TEN
00440 H0$(9)="5=AB[":H1$(9)="ZCDE6":rem ...JACK
00450 H0$(10)="7FGH[":H1$(10)="ZIJK8":rem .QUEEN
00460 H0$(11)="9FLM[":H1$(11)="ZINO:":rem .KING
00470 H0$(12)=";FxH[":H1$(12)="ZIyK<":rem .ACE

00480 rem ALLOCATE CLUBS TO STRINGS
00490 C0$(0)="#=g=?":C1$(0)="@>k>$":rem ...DEUCE
00500 C0$(1)="%=h=?":C1$(1)="@>l>&":rem ...THREE
00510 C0$(2)="'g=g?":C1$(2)="@k>k(":rem ...FOUR
00520 C0$(3)=")gfg?":C1$(3)="@kjk*":rem ...FIVE
00530 C0$(4)="+h=h?":C1$(4)="@l>l,":rem ...SIX
00540 C0$(5)="-hfh?":C1$(5)="@ljl.":rem ...SEVEN
00550 C0$(6)="/i=i?":C1$(6)="@m>m0":rem ...EIGHT
00560 C0$(7)="1ifi?":C1$(7)="@mjm2":rem ...NINE
00570 C0$(8)="3igi?":C1$(8)="@mkm4":rem ...TEN
00580 C0$(9)="5=ABd":C1$(9)="eCDE6":rem ...JACK
00590 C0$(10)="7FGHd":C1$(10)="eIJK8":rem .QUEEN
00600 C0$(11)="9FLMd":C1$(11)="eINO:":rem .KING
00610 C0$(12)=";FxHd":C1$(12)="eIyK<":rem .ACE

00620 rem ALLOCATE SPADES TO STRINGS
00630 S0$(0)="#=q=?":S1$(0)="@>u>$":rem ...DEUCE
00640 S0$(1)="%=r=?":S1$(1)="@>v>&":rem ...THREE
00650 S0$(2)="'q=q?":S1$(2)="@u>u(":rem ...FOUR
00660 S0$(3)=")qpq?":S1$(3)="@utu*":rem ...FIVE
00670 S0$(4)="+r=r?":S1$(4)="@v>v,":rem ...SIX
00680 S0$(5)="-rpr?":S1$(5)="@vtv.":rem ...SEVEN
00690 S0$(6)="/s=s?":S1$(6)="@w>w0":rem ...EIGHT
00700 S0$(7)="1sps?":S1$(7)="@wtw2":rem ...NINE
00710 S0$(8)="3sqs?":S1$(8)="@wuw4":rem ...TEN
00720 S0$(9)="5=ABn":S1$(9)="oCDE6":rem ...JACK
00730 S0$(10)="7FGHn":S1$(10)="oIJK8":rem .QUEEN
00740 S0$(11)="9FLMn":S1$(11)="oINO:":rem .KING
00750 S0$(12)=";FxHn":S1$(12)="oIyK<":rem .ACE

00760 rem ESTABLISH GRAPHICS IN PCG MEMORY
00770 P=-2048+35*16
00780 for I=P to P+(16*87)-1
00790 read D:poke I,D
00800 next I
00810 cls:pcg
```

```
00820 rem DIAMONDS
00830 curs 2,2 :printD0$(0):curs 2,3:printD1$(0)
00840 curs 8,2 :printD0$(1):curs 8,3:printD1$(1)
00850 curs 14,2:printD0$(2):curs 14,3:printD1$(2)
00860 curs 20,2:printD0$(3):curs 20,3:printD1$(3)
00870 curs 26,2:printD0$(4):curs 26,3:printD1$(4)
00880 curs 32,2:printD0$(5):curs 32,3:printD1$(5)
00890 curs 38,2:printD0$(6):curs 38,3:printD1$(6)
00900 curs 44,2:printD0$(7):curs 44,3:printD1$(7)
00910 curs 50,2:printD0$(8):curs 50,3:printD1$(8)
00920 curs 2,5 :printD0$(9):curs 2,6:printD1$(9)
00930 curs 8,5 :printD0$(10):curs 8,6:printD1$(10)
00940 curs 14,5:printD0$(11):curs 14,6:printD1$(11)
00950 curs 20,5:printD0$(12):curs 20,6:printD1$(12)

00960 rem PRINT HEARTS
00970 curs 2,8  :printH0$(0):curs 2,9:printH1$(0)
00980 curs 8,8  :printH0$(1):curs 8,9:printH1$(1)
00990 curs 14,8 :printH0$(2):curs 14,9:printH1$(2)
01000 curs 20,8 :printH0$(3):curs 20,9:printH1$(3)
01010 curs 26,8 :printH0$(4):curs 26,9:printH1$(4)
01020 curs 32,8 :printH0$(5):curs 32,9:printH1$(5)
01030 curs 38,8 :printH0$(6):curs 38,9:printH1$(6)
01040 curs 44,8 :printH0$(7):curs 44,9:printH1$(7)
01050 curs 50,8 :printH0$(8):curs 50,9:printH1$(8)
01060 curs 56,8 :printH0$(9):curs 56,9:printH1$(9)
01070 curs 2,11 :printH0$(10):curs 2,12:printH1$(10)
01080 curs 8,11 :printH0$(11):curs 8,12:printH1$(11)
01090 curs 14,11:printH0$(12):curs 14,12:printH1$(12)
01100 print:normal:print"PRESS ANY KEY FOR THE OTHER SUITS":pcg
01110 X1$=key:ifX1$=""then1110

01120 rem  CLUBS
01130 cls:curs 2,2:printC0$(0):curs 2,3:printC1$(0)
01140 curs 8,2 :printC0$(1):curs 8,3:printC1$(1)
01150 curs 14,2:printC0$(2):curs 14,3:printC1$(2)
01160 curs 20,2:printC0$(3):curs 20,3:printC1$(3)
01170 curs 26,2:printC0$(4):curs 26,3:printC1$(4)
01180 curs 32,2:printC0$(5):curs 32,3:printC1$(5)
01190 curs 38,2:printC0$(6):curs 38,3:printC1$(6)
01200 curs 44,2:printC0$(7):curs 44,3:printC1$(7)
01210 curs 50,2:printC0$(8):curs 50,3:printC1$(8)
01220 curs 56,2:printC0$(9):curs 56,3:printC1$(9)
01230 curs 2,5 :printC0$(10):curs 2,6:printC1$(10)
01240 curs 8,5 :printC0$(11):curs 8,6:printC1$(11)
01250 curs 14,5:printC0$(12):curs 14,6:printC1$(12)

01260 rem  SPADES
01270 curs 2,8  :printS0$(0):curs 2,9:printS1$(0)
01280 curs 8,8  :printS0$(1):curs 8,9:printS1$(1)
01290 curs 14,8 :printS0$(2):curs 14,9:printS1$(2)
01300 curs 20,8 :printS0$(3):curs 20,9:printS1$(3)
01310 curs 26,8 :printS0$(4):curs 26,9:printS1$(4)
01320 curs 32,8 :printS0$(5):curs 32,9:printS1$(5)
01330 curs 38,8 :printS0$(6):curs 38,9:printS1$(6)
```

```
01340 curs 44,8 :print$0$(7):curs 44,9:print$1$(7)
01350 curs 50,8 :print$0$(8):curs 50,9:print$1$(8)
01360 curs 2,11 :print$0$(9):curs 2,12:print$1$(9)
01370 curs 8,11 :print$0$(10):curs 8,12:print$1$(10)
01380 curs 14,11:print$0$(11):curs 14,12:print$1$(11)
01390 curs 20,11:print$0$(12):curs 20,12:print$1$(12)
01400 normal:end

01410 rem ---DATA COMMON TO ALL SUITS----
01420 data 255,190,130,130,190,160,160,190,128,128,128,128,128,
      128,128,128
01430 data 1,1,1,1,1,1,1,125,5,5,125,65,65,125,1,255
01440 data 255,128,188,132,132,188,132,132,188,128,128,128,128,
      128,128,128
01450 data 1,1,1,1,1,1,1,61,5,5,61,5,5,61,1,255
01460 data 255,128,160,160,168,188,144,144,128,128,128,128,128,
      128,128,128
01470 data 1,1,1,1,1,1,1,33,33,41,41,61,9,9,1,255
01480 data 255,128,188,160,160,188,132,132,188,128,128,128,128,
      128,128,128
01490 data 1,1,1,1,1,1,1,41,33,33,41,5,5,41,1,255
01500 data 255,128,188,160,160,188,164,164,188,128,128,128,128,
      128,128,128
01510 data 1,1,1,1,1,1,1,61,33,33,61,37,37,61,1,255
01520 data 255,128,188,132,132,136,136,144,128,128,128,128,128,
      128,128,128
01530 data 1,1,1,1,1,1,1,1,61,5,5,9,9,17,1,255
01540 data 255,128,188,164,164,188,164,164,188,128,128,128,128,
      128,128,128
01550 data 1,1,1,1,1,1,1,61,37,37,61,37,37,61,1,255
01560 data 255,128,188,164,164,188,132,132,188,128,128,128,128,
      128,128,128
01570 data 1,1,1,1,1,1,1,61,37,37,61,5,5,61,1,255
01580 data 255,128,175,169,169,169,169,169,175,128,128,128,128,
      128,128,128
01590 data 1,1,1,1,1,1,1,189,165,165,165,165,165,189,1,255
01600 data 255,128,159,132,132,132,132,132,200,176,128,128,128,
      128,128,128
01610 data 1,1,1,1,1,1,125,17,17,17,17,17,145,241,1,255
01620 data 255,128,190,161,161,161,161,169,165,190,130,129,128,
      128,128,128
01630 data 1,1,1,1,1,125,69,69,69,69,77,125,9,9,1,255
01640 data 255,128,162,164,168,180,164,162,128,128,128,128,128,
      128,128,128
01650 data 1,1,1,1,1,1,1,1,69,73,81,97,73,69,1,255
01660 data 255,128,190,162,162,162,190,162,162,128,128,128,128,
      128,128,128
01670 data 1,1,1,1,1,1,1,125,69,69,69,125,69,69,1,255
01680 data 255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
01690 data 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255
01700 data 255,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
01710 data 128,128,128,128,128,128,128,128,128,128,128,128,128,
      128,128,255
```

```
01720 rem --- DATA FOR COURT CARDS ---(COMMON)---
01730 data 255,0,0,0,0,0,0,255,255,255,255,255,15,15,15,15
01740 data 255,0,0,0,0,0,0,255,255,255,255,255,240,240,240,240
01750 data 0,0,0,0,255,255,255,255,255,255,0,0,0,0,0,255
01760 data 15,15,15,15,15,15,255,255,255,255,0,0,0,0,0,255
01770 data 240,240,240,240,240,240,240,240,240,240,0,0,0,0,0,255
01780 data 255,0,0,0,0,0,127,127,127,127,127,127,127,127,127,127
01790 data 255,0,0,0,0,0,255,255,255,255,129,129,129,129,129,129
01800 data 255,0,0,0,0,0,254,254,254,254,254,254,254,254,254,254
01810 data 127,127,127,127,127,127,127,127,127,127,0,0,0,0,0,255
01820 data 129,129,128,135,131,251,251,251,251,251,7,7,7,0,0,255
01830 data 254,254,126,126,126,126,126,126,126,126,0,0,0,0,0,255
01840 data 255,0,0,0,0,0,1,1,3,3,7,7,14,12,24,16
01850 data 255,0,0,0,0,0,252,248,240,224,192,128,0,0,0,0
01860 data 16,24,12,14,7,7,3,3,1,1,0,0,0,0,0,255
01870 data 0,0,0,0,128,192,224,240,248,252,0,0,0,0,0,255

01880 rem --- DATA FOR DIAMONDS ---
01890 data 128,128,128,128,128,128,128,128,128,144,184,252,184,
      144,128,255
01900 data 255,1,1,17,57,125,57,17,1,1,1,1,1,1,1,1
01910 data 255,0,0,0,0,0,0,0,0,0,0,0,0,16,56,124
01920 data 255,0,0,0,0,0,0,16,56,124,124,56,16,0,0,0,0
01930 data 255,0,0,0,16,56,124,124,56,16,0,0,0,16,56,124
01940 data 255,0,0,16,56,124,56,16,0,0,16,56,124,56,16,0
01950 data 124,56,16,0,0,0,0,0,0,0,0,0,0,0,0,255
01960 data 0,0,0,0,0,16,56,124,124,56,16,0,0,0,0,255
01970 data 124,56,16,0,0,0,16,56,124,124,56,16,0,0,0,255
01980 data 0,16,56,124,56,16,0,0,16,56,124,56,16,0,0,255

01990 rem --- DATA FOR HEARTS ---
02000 data 128,128,128,128,128,128,128,128,230,255,254,184,144,
      128,128,255
02010 data 255,1,1,103,255,127,57,17,1,1,1,1,1,1,1,1
02020 data 255,0,0,0,0,0,0,0,0,0,0,0,0,0,102,255
02030 data 255,0,0,0,0,0,102,255,126,56,16,0,0,0,0,0
02040 data 255,0,0,0,0,102,255,126,56,16,0,0,0,0,102,255
02050 data 255,0,0,102,255,126,56,16,0,0,102,255,126,56,16,0
02060 data 126,56,16,0,0,0,0,0,0,0,0,0,0,0,0,255
02070 data 0,0,0,0,0,102,255,126,56,16,0,0,0,0,0,255
02080 data 126,56,16,0,0,0,102,255,126,56,16,0,0,0,0,255
02090 data 0,102,255,126,56,16,0,0,102,255,126,56,16,0,0,255

02100 rem --- DATA FOR CLUBS ---
02110 data 255,1,1,25,61,25,103,255,91,25,1,1,1,1,1,1
02120 data 128,128,128,128,128,128,152,188,152,230,255,218,152,
      128,128,255
02130 data 255,0,0,0,0,0,0,0,0,0,0,0,0,24,60,24,102
02140 data 255,0,0,0,0,24,60,24,102,255,90,24,0,0,0,0
02150 data 255,0,0,24,60,24,102,255,90,24,0,0,24,60,24,102
02160 data 255,24,60,24,102,255,90,24,0,24,60,24,102,255,90,24
02170 data 255,90,24,0,0,0,0,0,0,0,0,0,0,0,0,255
02180 data 0,0,0,0,24,60,24,102,255,90,24,0,0,0,0,255
02190 data 255,90,24,0,0,24,60,24,102,255,90,24,0,0,0,255
02200 data 24,60,24,102,255,90,24,0,24,60,24,102,255,90,24,255
```

```
02210 rem --- DATA FOR SPADES ---
02220 data 255,1,1,25,61,127,255,229,25,1,1,1,1,1,1,1
02230 data 128,128,128,128,128,128,128,152,188,254,255,219,152,
      128,128,255
02240 data 255,0,0,0,0,0,0,0,0,0,0,0,0,24,60,126
02250 data 255,0,0,0,0,0,24,60,126,255,219,24,0,0,0,0
02260 data 255,0,0,0,24,60,126,255,219,24,0,0,0,24,60,126
02270 data 255,0,24,60,126,255,219,24,0,24,60,126,255,219,24,0
02280 data 255,219,24,0,0,0,0,0,0,0,0,0,0,0,0,255
02290 data 0,0,0,0,24,60,126,255,219,24,0,0,0,0,0,255
02300 data 255,219,24,0,0,0,24,60,126,255,219,24,0,0,0,255
02310 data 0,24,60,126,255,219,24,0,24,60,126,255,219,24,0,255
02320 data 255,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0
02330 data 255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,255
```

MACHINE LANGUAGE GAME TUTORIAL
... with sound and graphics

Written by D.S. Long 1984

By manipulating the paddle with "<" and ">",
rebound the ball to knock out bricks

POINTS SCORE

ROW 1:1 ROW 2:2 ROW 3:3 ROW 4:4 ROW 5:5 ROW 6:10

Press any key to continue.

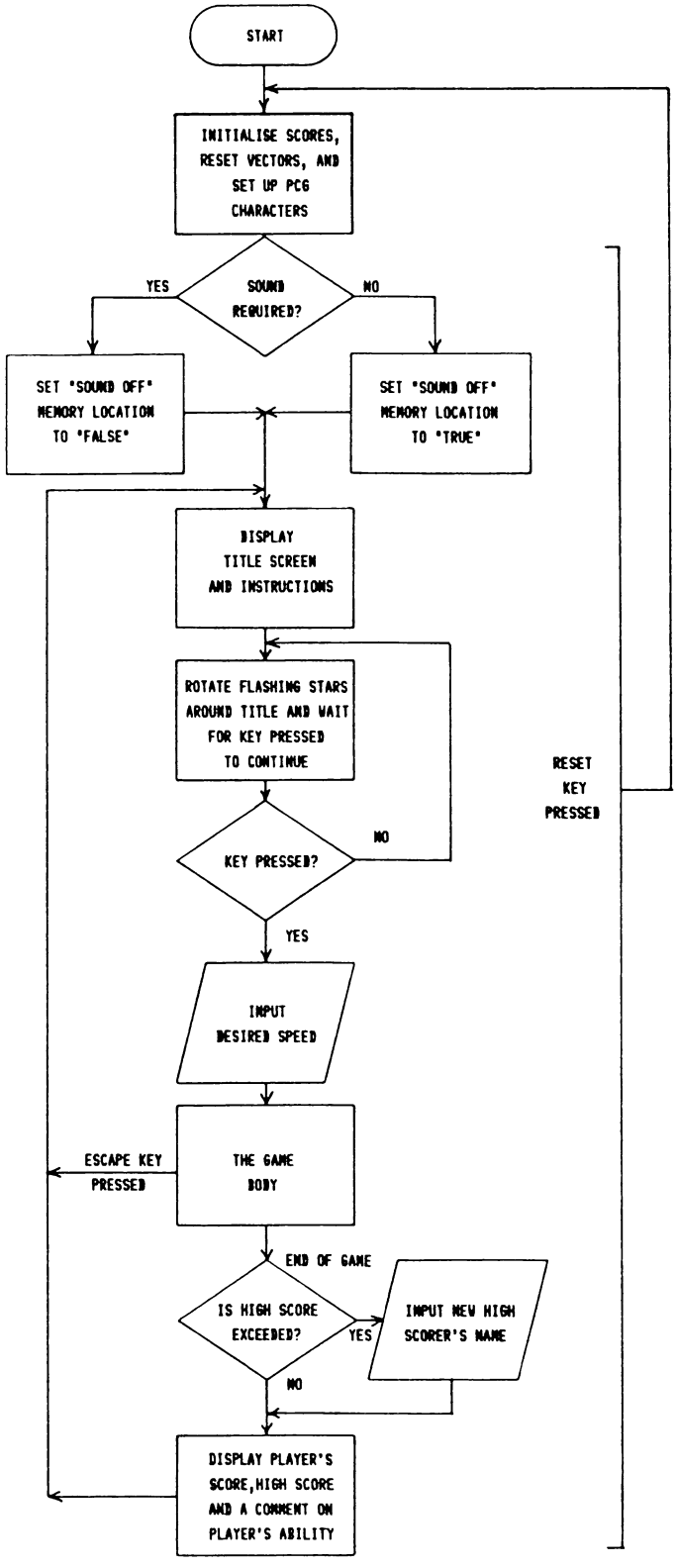
INTRODUCTION

This chapter contains a program which has been listed as an EDASM source file. It is presented as a tutorial to show the machine code novice that programming the Microbee's Z-80 chip is not as daunting a prospect as it may first appear. The program was written with only basic programming experience in high level languages (BASIC, FORTRAN, PASCAL) and none at all in machine code!

Brick-ball is a game program in which the player, by manipulating a paddle to the left and right with either the '<' and '>' keys or a joystick, rebounds a bouncing ball into a wall of bricks to knock the bricks out one by one. Bricks knocked out score points which increase towards the back of the wall (top of the screen). Three misses of the ball with the paddle are normally allowed, but an extra ball is granted when a score of 400 points is reached. The ball's speed is selectable at the start of the game, but when the 'back wall' is hit for the first time in each frame, this speed will increase slightly. At the end of the game there is a facility for a player to enter his name against a new high score (originally held by 'Microbee' for a score of 100) and the high score and player's score are displayed. A comment on the player's ability is also made, according to score.

The program also makes use of the Microbee's built-in speaker. A range of tunes and sounds are produced at various stages in the game.

The flow chart for the game, as the player sees it, is as follows:



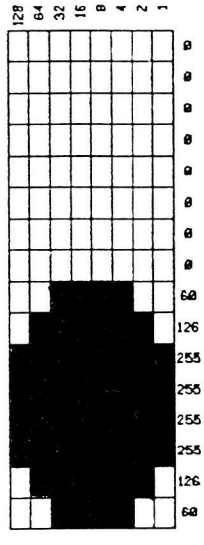
The following points will be of interest to the prospective machine code games programmer:

PCG CHARACTERS

PCG characters are easy to set up and are essential to give your game good graphic representation.

The illustration below shows the character described by Brick-ball's first 16 PCG data values (found under the label PCGDAT) - the ball printed at the bottom of a character space. Each data value describes a row of the character's pixels, a pixel being one of the 128 character subdivisions (see the figures to the right of the illustration). These values are determined by deciding which of the pixels are to be set, then adding across the row the value at the top of the column corresponding to each square set. PCG character design programs are available or can easily be written to save effort.

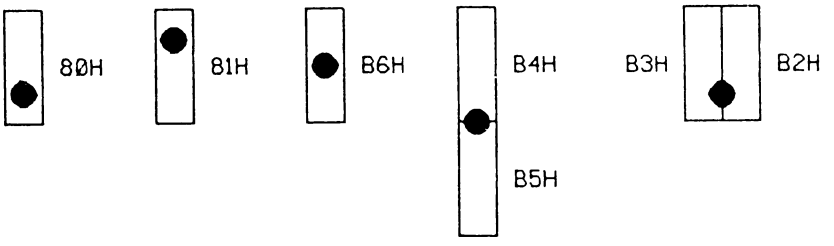
Once designed, the character's data must be stored in the PCG RAM (F8ØØ - FFFFH). The first 16 PCG RAM memory locations describe the character accessed by 8ØH, the next 16 81H, etc. For instance, if the sequence for the character below were stored in locations F8ØØ - F8ØFH, and 8ØH stored at the first screen RAM location (FØØØH), the character would be printed in the top lefthand corner of the screen.



SCREEN PLANNING AND CHARACTER MANIPULATION

The easiest way to plan what is wanted on the screen where and when is to draw up a character screen map as shown at Appendix Four. The VDU screen contains 1Ø24 characters (64 wide by 16 high) numbered from FØØØH to F3FFH as shown at Appendix Four.

By noting on the map where various characters are to go, it is then simple to store the code of those characters in the appropriate screen RAM locations at the appropriate time. Brick-ball uses a simple method of character manipulation by defining beforehand all the characters to be used in the game, e.g. for the ball:



By storing the hex. code of these characters in the appropriate screen RAM, the ball can be made to 'move' around the screen.

A method by which more fluid apparent movement may be displayed is to have the program redesign the PCGs according to need and position. This method, although giving more satisfying results, is difficult.

LOADING THE PROGRAM

The following EDASM listing is designed for use in the 64k, 32kIC or the older 2MHz 32k MicroBees, with or without disk drive. Slight variations are necessary for the different types of machine being used, which are described in detail at the appropriate points. For example, PCG alphanumerics seen at pages 87/88 at the top of the screen have been omitted in the 32k version, and ASCII used instead. It has been assumed throughout that the owner knows how to operate EDASM and save the assembled program.

The file is well-commented (anything following a semi-colon on a line is a comment), although comments must not be entered to allow the file to fit into the memory available. Descriptions of program sections are also interspersed with the code.

Users of a 64k Bee with drive should answer the 'Memory Size?' question (when EDASM is first called) with B000H, and those without a drive or 32k users should allow 7FFFH. Users of a 64k Bee will probably wish to create a .COM type file. This can easily be done by changing STRTAD in line 8 to 0100H and assembling with an offset. The program has been designed to be totally independent of BASIC or the 64k Bee's Boot ROM: it contains its own key scanning routines and sets its own screen format on the 6545 driver. WARNING: The code must be entered exactly as shown, including spaces, or else the program may not run correctly!

Program name : BRKBAL

```
00001   ;*************************************************************************
00002   ;       BRICKBALL   by D.S. Long for WILDCARDS - 16 June 1984
00003   ;*************************************************************************
```

The equivalence table assigns labels to numerical values often used within the program. It is included as part of good programming technique, ensuring that the code will not consist of a string of unintelligible numbers, but of identifiable labels.

```
00004
00005   FALSE   EQU     0       ;value of false
00006   TRUE    EQU     1       ;value of true
00007   RSTADD  EQU     00A2H   ;reset vector for BASIC
00008   STRTAD  EQU     0900H   ;autostart address - use 0100H for disk MicroBee
00009   SPKROF  EQU     184     ;speaker port low output
00010   SPKRON  EQU     248     ;speaker port high output
00011   PSTEST  EQU     0DF7BH  ;BOOTROM reset vector = 55AAH if next two bytes hold
00012                           ;       address to be executed after a reset
00013   VDU     EQU     0F000H  ;start of screen RAM
00014   SCORPS  EQU     0F00DH  ;position where score is printed in game
00015   FRMPOS  EQU     0F021H  ;position where frame no. is printed in game
00016   MENPOS  EQU     0F03AH  ;position where no. of balls left is printed
00017   BALLIN  EQU     0F201H  ;number added to random no. for ball start position
00018   STPAD   EQU     0F3DCH  ;initial position of lefthand end of paddle
00019   PCGRAM  EQU     0F800H  ;start of PCG RAM
00020
```

The program body begins by setting the default radix to 10 (decimal), i.e., all numbers used from here on will be in decimal unless appended otherwise. The assembler is then instructed to compile the program at 'STRTAD'. From there, all necessary initialization for the program to run takes place, including setting 64*16 screen, initializing joystick port, setting up inverse PCG characters, resetting BASIC and Boot ROM reset vectors and setting up the PCG characters to be used in the program. Do not be alarmed at the mention of the Boot ROM if you have a 32k Bee - the program merely restarts when the RESET key is pressed.

```
00024           DEFR    10D
00025           ORG     STRTAD          ;origin of program at STRTAD
00026   ASTART  LD      HL,D6545        ;6545 data to set 64*16 screen format
00027           LD      B,16
00028   LP6545  LD      A,B             ;loop to set 64*16 screen
00029           DEC     A
00030           OUT     (0CH),A
00031           LD      A,(HL)
00032           OUT     (0DH),A
00033           DEC     HL
00034           DJNZ    LP6545
00035           LD      A,0FFH
```

```
00036           OUT     (01),A          ;initialize port for joystick
00037           LD      HL,3000H
00038   DLYLP   DEC     HL              ;delay loop to wait for 6545 adjustments
00039           LD      A,H
00040           OR      L
00041           JR      NZ,DLYLP
00042           LD      A,1
00043           OUT     (0BH),A         ;select ROM read latch on
00044           LD      SP,STKPTR       ;set up user defined stack
00045           LD      HL,VDU          ;start of inverse character data in ROM
00046           LD      DE,PCGRAM
00047           LD      BC,800H
00048   CHLOOP  LD      A,(HL)          ;loop to change data to inverse data
00049           CPL                     ;take complement
00050           LD      (DE),A          ;load inverse character data into PCG RAM
00051           INC     HL
00052           INC     DE
00053           DEC     BC
00054           LD      A,B
00055           OR      C
00056           JR      NZ,CHLOOP       ;return for next character
00057           LD      A,0
00058           OUT     (0BH),A         ;select ROM read latch off
00059           LD      HL,ASTART
00060           LD      (RSTADD),HL     ;set basic reset vector to autostart address
00061           LD      (PSTEST+2),HL   ;set BOOTROM reset vector
00062           LD      HL,0AA55H
00063           LD      (PSTEST),HL     ;set BOOTROM "reset vector set indicator"
00064           LD      HL,PCGDAT
00065           LD      DE,PCGRAM
00066           LD      BC,912
00067           LDIR                    ;set up PCG characters
```

The screen is now cleared and the first screen message, 'DO YOU WANT SOUND (Y/N)?', printed. The selected option is recorded.

```
00068           CALL    CLS             ;subroutine to clear screen
00069           LD      HL,SCRN18
00070           LD      DE,0F1D4H
00071           LD      BC,24
00072           LDIR                    ;print "select sound" message
00073   SNDLP3  LD      A,0EH           ;keyboard code for "N" key
00074           CALL    TESTKY          ;test for key (code in A register) pressed
00075           JR      Z,SNDLP1        ;jump if "N" key pressed
00076           LD      A,19H           ;keyboard code for "Y" key
00077           CALL    TESTKY
00078           JR      Z,SNDLP2
00079           JR      SNDLP3          ;if no key pressed, try again
00080   SNDLP1  LD      A,TRUE
00081           LD      (SNDOFF),A      ;set "sound off" flag to TRUE
00082           JR      NEWGME
00083   SNDLP2  LD      A,FALSE
00084           LD      (SNDOFF),A      ;set "sound off" flag to FALSE
```

Initialization of variables used within the game now takes place (SCORE2 = total score of player, MEN = no. of balls remaining, LSTHUN = record of whether extra ball score has been reached, FRAME = current frame number) and the title screen with game instructions printed. The tune for the start of the game is also played here.

The title screen also includes the instruction, 'Press any key to continue.', hence the 'INITWT' delay. If it were not included, during the period of depression of the sound option key, the title screen would be printed and continue on, as a key is already depressed.

```
00085  NEWGME  LD      DE,0            ;return address for new game
00086          LD      (SCORE2),DE     ;reset score
00087          LD      A,3
00088          LD      (MEN),A         ;reset ball counter
00089          LD      A,0
00090          LD      (LSTHUN),A      ;reset counter for extra ball
00091          LD      A,0
00092          LD      (FRAME),A       ;reset frame counter
00093          CALL    CLS
00094          LD      HL,SCRN04       ;top half of word "BRICK-BALL" in title frame
00095          LD      DE,0F0CDH       ;screen location
00096          LD      BC,37           ;take 37 PCG characters
00097          LDIR
00098          LD      HL,SCRN05       ;rest of word "BRICK-BALL" in title frame
00099          LD      DE,0F10DH
00100          LD      BC,37
00101          LDIR
00102          LD      HL,SCRN06       ;"Written by D.S. Long....." etc
00103          LD      DE,0F193H
00104          LD      BC,25
00105          LDIR
00106          LD      HL,SCRN07       ;"By manipulating....." etc
00107          LD      DE,0F24AH
00108          LD      BC,44
00109          LDIR
00110          LD      HL,SCRN08       ;"rebound....." etc
00111          LD      DE,0F28EH
00112          LD      BC,36
00113          LDIR
00114          LD      HL,SCRN09       ;"POINTS SCORE" (in Inverse)
00115          LD      DE,0F319H
00116          LD      BC,14
00117          LDIR
00118          LD      HL,SCRN0A       ;"ROW 1:1....." etc
00119          LD      DE,0F343H
00120          LD      BC,58
00121          LDIR
00122          LD      HL,SCRN0B       ;"Press any....." etc
00123          LD      DE,0F3D3H
00124          LD      BC,26
00125          LDIR                    ;end loading title screen
00126          LD      E,250           ;load E with time low for speaker port
```

```
00127           LD      HL,ROVDAT       ;start of first tune data
00128           LD      A,(SNDOFF)
00129           OR      A
00130           CALL    Z,PLYTUN        ;call subroutine to play tune if required
00131           LD      HL,8000H        ;delay to stop key inputs overlapping
00132   INITWT  LD      A,H
00133           OR      L
00134           DEC     HL
00135           JR      NZ,INITWT
```

The next section prints a box of stars around the game title and rotates them. Between each quanta of rotation, the random number generator is randomizing and a key input for the game to continue is searched for.

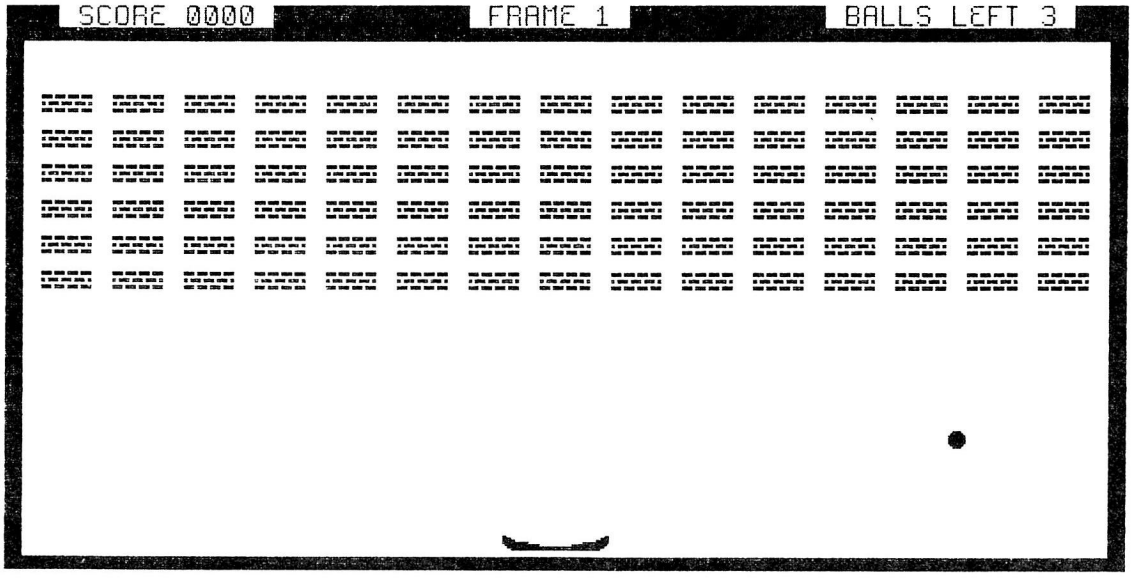
```
00136           LD      DE,202AH        ;load E with ASCII value of a star
00137           LD      BC,2020H        ;load C with a blank
00138   LOOP03  LD      HL,0F08BH       ;first star position
00139   LOOP04  LD      (HL),E          ;print a star
00140           INC     HL
00141           LD      (HL),C          ;print a blank
00142           INC     HL
00143           LD      A,L
00144           CP      0B3H
00145           JR      NZ,LOOP04       ;continue to print top row of stars
00146           LD      (HL),E          ;print right wall of stars
00147           LD      (0F0F3H),DE     ;screen positions for R/H wall
00148           LD      (0F133H),DE     ;         ditto
00149           LD      (0F173H),DE     ;         ditto
00150           LD      (0F1B3H),DE     ;         ditto
00151           LD      HL,0F1F3H
00152   LOOP05  LD      (HL),C          ;loop to print bottom row of stars
00153           DEC     HL
00154           LD      (HL),E
00155           DEC     HL
00156           LD      A,L
00157           CP      0CBH
00158           JR      NZ,LOOP05
00159           LD      (HL),C          ;print left wall of stars
00160           LD      (0F18BH),DE
00161           LD      (0F14BH),DE
00162           LD      (0F10BH),DE
00163           LD      (0F0CBH),DE
00164           PUSH    DE
00165           PUSH    BC
00166           POP     DE              ;swap star for blank
00167           LD      A,10H
00168           OUT     (0CH),A
00169           IN      A,(0DH)         ;initialize port for keyboard input
00170           LD      BC,3FFFH        ;delay in star rotation
00171   LOOP06  DEC     BC
00172           LD      HL,RANNUM       ;begin generating random number
00173           INC     (HL)
00174           IN      A,(0CH)
00175           BIT     6,A             ;check for any key pressed
```

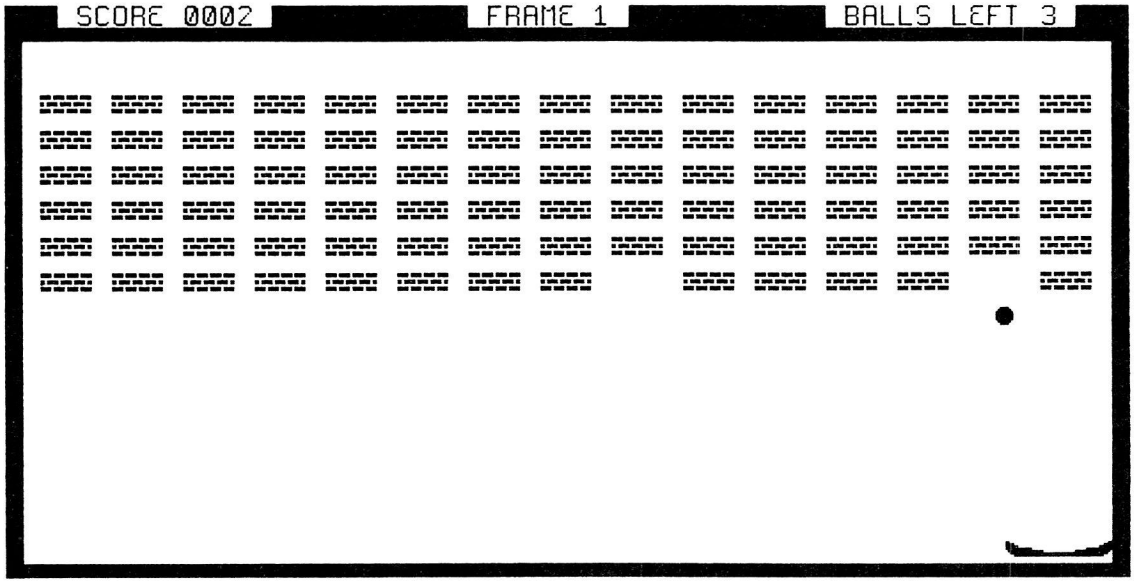
```
00176           JR      NZ,GAMEST       ;if key pressed then next stage
00177           LD      A,C
00178           OR      B
00179           JR      NZ,LOOP06
00180           POP     BC
00181           JR      LOOP03          ;return to move stars again
```

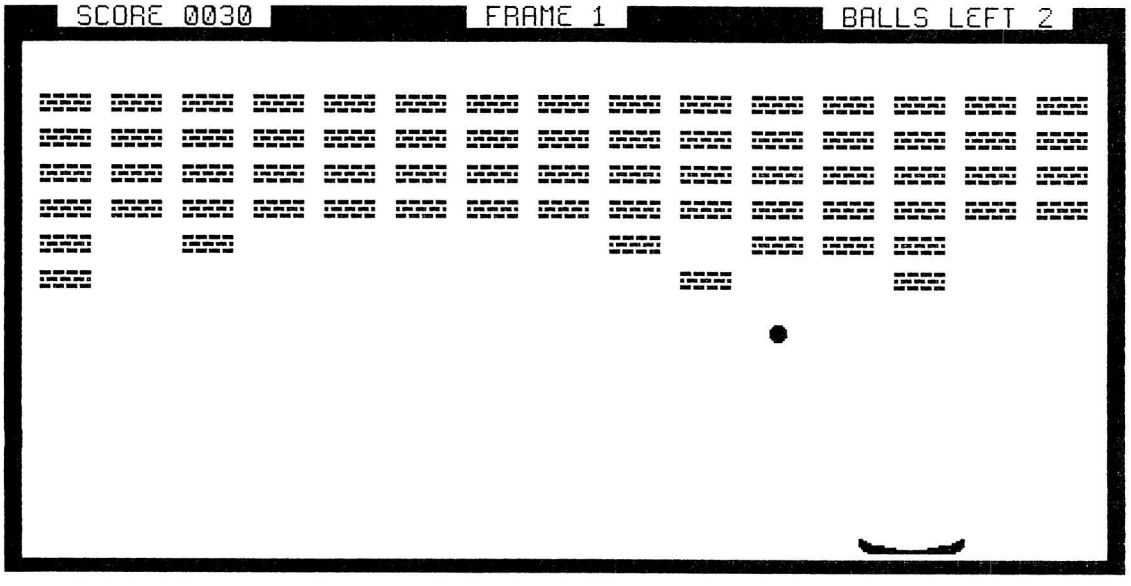
Speed of ball movement is now selected. Three speeds are available (1 = fast, 2 = medium, 3 = slow).

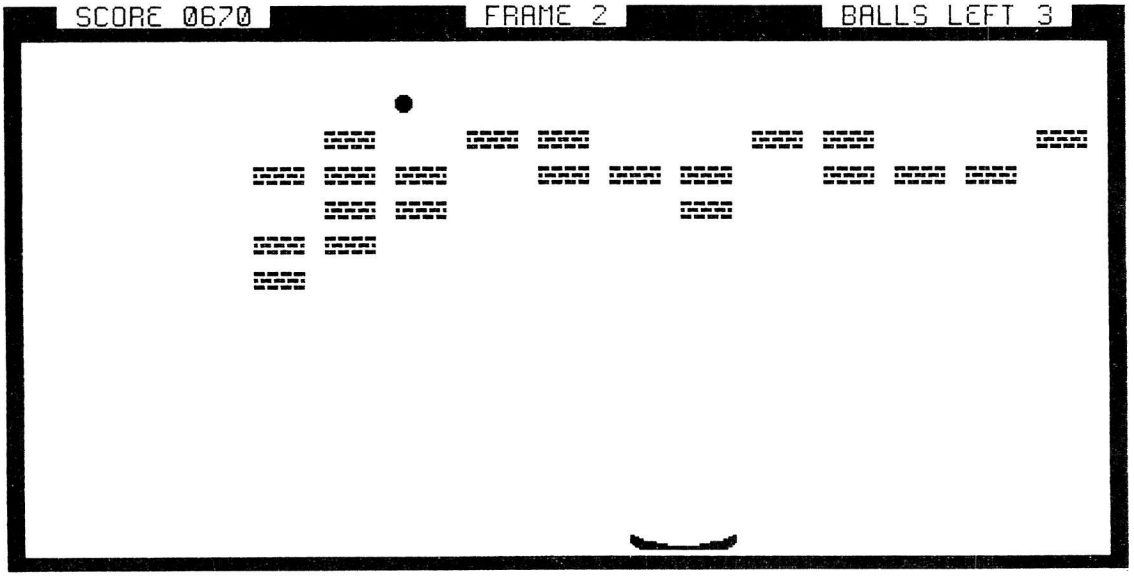
```
00182   GAMEST  POP     BC
00183
00184           CALL    CLS
00185           LD      HL,SCRN16
00186           LD      DE,0F1D0H
00187           LD      BC,33
00188           LDIR                    ;print "select speed" message
00189   SPDLOP  LD      A,33            ;speed setting routine
00190           LD      BC,600H         ;high speed delay (0300H for 2 MHz Bees)
00191           LD      (DELAY),BC      ;speed set by searching for
00192           CALL    TESTKY          ;       key pressed corresponding to
00193           JR      Z,WSTART        ;       particular speed
00194           LD      A,34
00195           LD      BC,900H         ;medium speed delay (0450H for 2 MHz Bees)
00196           LD      (DELAY),BC
00197           CALL    TESTKY
00198           JR      Z,WSTART
00199           LD      A,35
00200           LD      BC,0C00H        ;low speed delay (0600H for 2 MHz Bees)
00201           LD      (DELAY),BC
00202           CALL    TESTKY
00203           JR      Z,WSTART
00204           JR      SPDLOP          ;if no key pressed, try again
```

Other necessary frame - to - frame initializations are now performed, the frame no. printed, and the game screen is printed.




SCORE 0002
FRAME 1
BALLS LEFT 3


SCORE 0030
FRAME 1
BALLS LEFT 2


SCORE 0670
FRAME 2
BALLS LEFT 3

```
00205  WSTART  LD      A,0             ;warm start address for new frame
00206          LD      (BRICKS),A      ;reset no. of bricks hit score to 0
00207          LD      HL,(DELAY)      ;adjust speed delay for use
00208          DEC     HL              ;       later in increasing speed
00209          LD      (DELAY),HL      ;       in each frame
00210          LD      DE,0
00211          LD      (SCORE1),DE     ;reset last points scored to zero
00212          CALL    CLS
00213          LD      HL,SCRN17
00214          LD      DE,0F1DCH
00215          LD      BC,5
00216          LDIR                    ;print "frame" message
00217          LD      A,(FRAME)
00218          ADD     A,31H
00219          LD      (0F1E2H),A      ;print frame no.
00220          LD      B,4
00221  FRMDEL  LD      DE,0CFFFH       ;delay whilst frame no. on screen
00222  FRMLOP  DEC     DE
00223          LD      A,D
00224          OR      E
00225          JR      NZ,FRMLOP
00226          DJNZ    FRMDEL
00227          LD      HL,SCRN01
00228          LD      DE,0F001H
00229          LD      BC,61
00230          LDIR                    ;print top line of game screen
00231          CALL    INITSC          ;print initial frame score
00232          LD      A,(FRAME)
00233          INC     A
00234          DAA                     ;increment no. of frames
00235          AND     0FH
00236          LD      (FRAME),A       ;print frame no.
```

IF YOU ARE USING 64K MICROBEE WITH DISK DRIVE THE NEXT LINES SHOULD READ:

```
00237          ADD     A,97H           ;PCG character offset
00238          LD      (FRMPOS),A      ;screen location
00239          LD      A,(MEN)         ;no. of balls remaining
00240          ADD     A,97H
```

ELSE THE NEXT LINES SHOULD READ:

```
00237          ADD     A,30H           ;ASCII character offset
00238          LD      (FRMPOS),A      ;screen location
00239          LD      A,(MEN)         ;no. of balls remaining
00240          ADD     A,30H
```

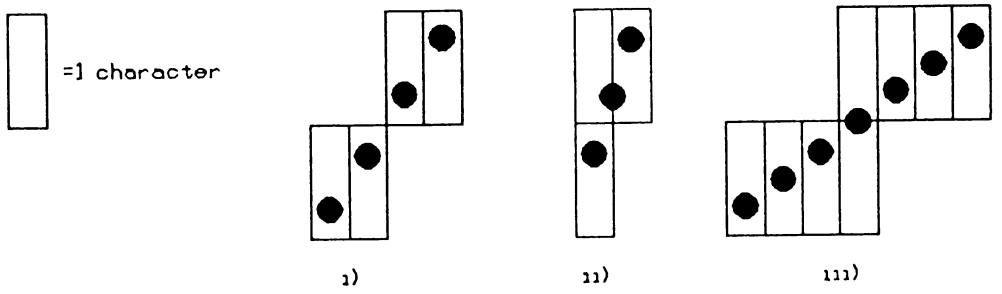
CONTINUE ENTERING AS NORMAL.....

```
00241          LD      (MENPOS),A      ;print no. of balls remaining
00242          LD      DE,0F082H       ;screen location of top LH brick
00243          LD      A,0
00244          PUSH    AF
00245  BKLOP1  LD      A,0             ;loop to print rows of bricks
```

```
00246   BKLOP2  LD      HL,BRICK        ;loop to print a row of bricks
00247           LD      BC,4            ;length of brick
00248           LDIR                    ;print a brick
00249           INC     A
00250           CP      15              ;have 15 bricks been printed?
00251           JR      NZ,BKLOP2       ;if not, return to print next brick
00252           POP     AF
00253           INC     A
00254           CP      6               ;have 6 rows of bricks been printed?
00255           JR      Z,BKLOP3        ;if so, then proceed to BKLOP3
00256           PUSH    AF
00257           EX      DE,HL
00258           LD      DE,4
00259           ADD     HL,DE           ;increase HL to start of next row
00260           EX      DE,HL
00261           JR      BKLOP1          ;return to print another row
00262   BKLOP3  LD      HL,VDU          ;begin printing borders
00263           LD      DE,40H
00264           LD      A,0
00265   BKLOP4  LD      (HL),0A4H       ;left border
00266           INC     A
00267           CP      16              ;height of screen = 16 rows
00268           JR      Z,BKLOP5
00269           ADD     HL,DE
00270           JR      BKLOP4
00271   BKLOP5  LD      HL,0F03EH
00272   BKLOP6  LD      (HL),0A4H       ;right border
00273           INC     A
00274           CP      32
00275           JR      Z,BKLOP7
00276           ADD     HL,DE
00277           JR      BKLOP6          ;end printing borders
00278   BKLOP7  LD      HL,0F3C1H
00279           LD      DE,0F3C2H
00280           LD      BC,60
00281           LD      (HL),88H
00282           LDIR                    ;print bottom of screen
00283           EXX
00284           LD      HL,SCRN03       ;paddle
00285           LD      DE,STPAD        ;set screen locn. of init. LH end of paddle
00286           PUSH    DE              ;store current paddle position
00287           LD      BC,6            ;length of paddle
00288           LDIR                    ;print paddle
00289           POP     DE              ;restore current paddle position
00290           EXX
```

The ball now begins its movement around the screen. There are three modes of ball movement possible, according to which section of the paddle the ball is rebounded from. These are:

i) Normal bounce.
ii) Steep bounce.
iii) Shallow bounce.



Register pair HL contains the current ball position; register pairs BC and DE contain the vertical and horizontal ball motion increments of +/-40H (64 columns) and +/-1 respectively (note that if any screen character position has 40H added to it, then it merely becomes the screen character position immediately below).

The ball is moved by first storing (PUSHing) HL and adding the horizontal increment to HL. The new screen position is then checked for being blank (20H). If it is not, then the ball must bounce. The original HL pair is then retrieved (POPped) and similar carried out for vertical and diagonal. When the new position has been thoroughly checked, the old ball is blanked and the new ball printed.

```
00291           LD      A,81H           ;(see diagram on page 82)
00292           EX      AF,AF'          ;load A' with ball type to be printed
00293   LOOP09  CALL    RANDOM          ;generate random number (0-61)
00294           LD      HL,0
00295           LD      L,A
00296           LD      DE,BALLIN
00297           ADD     HL,DE           ;load HL with ball start position
00298           LD      (HL),81H        ;       and print ball
00299           LD      A,0             ;load direction type of ball motion (0 =
00300           LD      (DIRTYP),A      ;       normal, 1 = shallow bounce)
00301           LD      A,1             ;initialize need for ball between print as
00302           LD      (BETNEC),A      ;       not necessary
00303           LD      B,0FH           ;delay to allow player to ready
00304   LOOP0A  CALL    PADLMV          ;move paddle
00305           DJNZ    LOOP0A
00306           CALL    RANDOM
00307           LD      BC,40H          ;load BC with vert. ball motion
00308           LD      DE,1            ;load DE with horizontal ball movement
00309           CP      30              ;       according to key pressed, either +1
00310           JR      C,LOOP07        ;       (right) or -1 (left), depending on
00311           LD      DE,-1           ;       random number
00312   LOOP07  JP      BALBET          ;loop to move ball in shallow and normal mode
00313   BBRET   EX      AF,AF'
00314           LD      (HL),A          ;print ball
00315           CP      81H             ;exchange ball printed in top of character
00316           JR      Z,BALLCH        ;       space (81H) for ball printed in
00317           INC     A               ;       bottom of character space (80H)
00318           INC     A               ;       or vice versa
00319   BALLCH  DEC     A
```

```
00320           EX      AF,AF'
00321           CALL    PADLMV
00322   EMSC01  PUSH    HL              ;store ball position
00323           ADD     HL,DE           ;add horizontal increment
00324           LD      A,(HL)          ;check for any non-space right/left of the
00325           CP      20H             ;       ball for it to bounce off
00326           CALL    NZ,BNCE01       ;if a non-space exists, then bounce
00327           POP     HL              ;restore ball position
00328           PUSH    HL
00329           CALL    ADDBC           ;add vertical increment if necessary
00330           OR      A
00331           JR      NZ,NOBC         ;if not necessary, then jump
00332           LD      A,(HL)          ;check for non-space above/below the ball
00333           CP      20H             ;       that it must bounce off
00334           JP      NZ,BNCE02       ;if non-space found, then bounce
00335   B02RET  POP     HL              ;subroutine BNCE02 return label
00336           PUSH    HL
00337           EX      AF,AF'
00338           PUSH    AF
00339           EX      AF,AF'
00340           POP     AF
00341           CP      80H             ;check if next ball to be printed is 80H
00342           JR      Z,NOBC          ;if it is, then jump
00343           ADD     HL,BC
00344           ADD     HL,DE
00345           LD      A,(HL)
00346           CP      94H             ;check for brick below and in direction of
00347           JP      Z,BNCE2C        ;       horizontal travel - if there is,
00348           CP      96H             ;       then bounce as for a corner
00349           JP      Z,BNCE2C        ;       (BNCE2C)
00350   NOBC    POP     HL
00351           PUSH    HL
00352           EX      AF,AF'
00353           PUSH    AF
00354           EX      AF,AF'
00355           POP     AF
00356           CP      81H
00357           JR      Z,NOCNR         ;no check for corner bounce required
00358           CALL    ADDBC
00359           ADD     HL,DE
00360           LD      A,(HL)
00361           CP      20H             ;check for non-space diagonally from ball
00362           CALL    NZ,BNCE03       ;if there is one, then bounce
00363   NOCNR   POP     HL
00364           PUSH    HL
00365           CALL    ADDBC
00366           ADD     HL,DE
00367           LD      A,(HL)          ;check that the new ball position found is
00368           CP      20H             ;       a space
00369           POP     HL
00370           JR      NZ,EMSC01       ;if not, go back and try again
00371           PUSH    HL              ;store old ball position
00372           CALL    ADDBC
00373           ADD     HL,DE           ;final new ball position calculated
00374           EXX
```

```
00375           POP     HL
00376           LD      A,(BETNEC)
00377           CP      0FFH            ;check that old ball is to be blanked
00378           JR      Z,NOBLNK
00379           LD      (HL),20H        ;if so, then blank it
00380   NOBLNK  EXX
00381           LD      A,(BRICKS)
00382           CP      5BH             ;check that all the bricks have not been hit
00383           JP      Z,WSTART        ;if so, then new frame
00384           JR      LOOP07          ;return to move ball again
00385   EMSC02  LD      A,B             ;loop to move ball in steep mode
00386           OR      A
00387           JR      Z,EMSC04
00388           LD      A,E
00389           CP      1               ;routine to print a ball halfway between
00390           JR      Z,HFBAL1        ;       the old ball position and the
00391           LD      A,0B3H          ;       new - once a new position has been
00392           LD      (HL),A          ;       selected, a ball is printed to
00393           DEC     A               ;       give movement continuity
00394           JR      HFBAL2
00395   HFBAL1  LD      A,0B2H
00396           LD      (HL),A
00397           INC     A               ;note only 81H ball used in EMSC02 routine
00398   HFBAL2  SBC     HL,DE
00399           LD      (HL),A
00400           CALL    PDLMV2
00401           LD      (HL),20H
00402           ADD     HL,DE
00403           JR      EMSC05
00404   EMSC04  SBC     HL,BC
00405           LD      A,E
00406           CP      1
00407           JR      Z,HFBAL3
00408           LD      A,0B3H
00409           LD      (HL),A
00410           DEC     A
00411           JR      HFBAL4
00412   HFBAL3  LD      A,0B2H
00413           LD      (HL),A
00414           INC     A
00415   HFBAL4  SBC     HL,DE
00416           LD      (HL),A
00417           CALL    PDLMV2
00418           LD      (HL),20H
00419           ADD     HL,DE
00420           LD      (HL),20H
00421           ADD     HL,BC           ;end of continuity ball routine
00422   EMSC05  LD      (HL),81H        ;print ball in new position, and so on as
00423           CALL    PDLMV2          ;       for LOOP07
00424           PUSH    HL
00425   B04RET  POP     HL
00426   EMSC03  PUSH    HL
00427           ADD     HL,DE
00428           LD      A,(HL)
00429           CP      20H
```

```
00430           JP      NZ,BNCE04
00431   B05RET  POP     HL
00432           PUSH    HL
00433           ADD     HL,BC
00434           LD      A,(HL)
00435           CP      20H
00436           JP      NZ,BNCE05
00437           POP     HL
00438           PUSH    HL
00439           ADD     HL,DE
00440           ADD     HL,BC
00441           LD      A,B
00442           OR      A
00443           JR      NZ,NBC
00444           LD      A,(HL)
00445           CP      94H
00446           JP      Z,BNCE5B
00447           CP      96H
00448           JP      Z,BNCE5B
00449   NBC     LD      A,(HL)
00450           CP      20H
00451           CALL    NZ,BNCE03
00452           POP     HL
00453           PUSH    HL
00454           ADD     HL,DE
00455           ADD     HL,BC
00456           LD      A,(HL)
00457           CP      20H
00458           POP     HL
00459           JR      NZ,EMSC03
00460           PUSH    HL
00461   B5CRET  POP     HL
00462           LD      (HL),20H
00463           ADD     HL,DE
00464           ADD     HL,BC
00465           LD      A,(BRICKS)
00466           CP      5BH
00467           JP      Z,WSTART
00468           JP      EMSC02          ;end of steep movement loop
00469   BALBET  LD      A,(DIRTYP)      ;routine to print intermediate ball in shallow mode
00470           OR      A
00471           JP      Z,BBRET         ;return if in normal mode
00472           LD      A,(BETNEC)      ;intermediate ball is only printed every
00473           CP      0FFH            ;       second ball, so if not needed this
00474           JR      Z,NEXTIM        ;       time, jump
00475           LD      A,B
00476           OR      A
00477           JR      Z,DOWNBL        ;if ball going down, then jump
00478           EX      AF,AF'
00479           CP      80H
00480           JR      NZ,UP2
00481           EX      AF,AF'
00482           LD      (HL),0B4H       ;print ball, delay, and remove it
00483           XOR     A
00484           SBC     HL,BC
```

```
00485           PUSH    HL
00486           LD      A,(HL)
00487           PUSH    AF
00488           CP      20H
00489           JR      Z,ONBRK1
00490           CP      96H
00491           JR      Z,ONBRK2
00492           LD      (HL),0B7H
00493           JR      ONBRK3
00494   ONBRK2  LD      (HL),0B8H
00495           JR      ONBRK3
00496   ONBRK1  LD      (HL),0B5H
00497   ONBRK3  CALL    PADLMV
00498           ADD     HL,BC
00499           LD      (HL),20H
00500           POP     AF
00501           POP     HL
00502           LD      (HL),A
00503           LD      A,0FFH
00504           LD      (BETNEC),A      ;select intermediate ball not to be printed next time
00505           JP      EMSC01
00506   UP2     EX      AF,AF'
00507           LD      (HL),0B6H
00508           CALL    PADLMV
00509           LD      (HL),20H
00510           LD      A,0FFH
00511           LD      (BETNEC),A
00512           JP      EMSC01
00513   DOWNBL  EX      AF,AF'
00514           CP      81H
00515           JR      NZ,UP2
00516           EX      AF,AF'
00517           LD      A,(HL)
00518           PUSH    AF
00519           CP      20H
00520           JR      Z,ONBRK5
00521           CP      94H
00522           JR      Z,ONBRK6
00523           LD      (HL),0B8H
00524           JR      ONBRK7
00525   ONBRK6  LD      (HL),0B7H
00526           JR      ONBRK7
00527   ONBRK5  LD      (HL),0B5H
00528   ONBRK7  XOR     A
00529           SBC     HL,BC
00530           POP     AF
00531           PUSH    HL
00532           PUSH    AF
00533           LD      (HL),0B4H
00534           CALL    PADLMV
00535           ADD     HL,BC
00536           POP     AF
00537           LD      (HL),A
00538           POP     HL
00539           LD      (HL),20H
```

```
00540           LD      A,0FFH
00541           LD      (BETNEC),A
00542           JP      EMSC01          ;end ball between (shallow) routine
00543   NEXTIM  LD      A,1
00544           LD      (BETNEC),A      ;select ball between needed next time
00545           JP      BBRET
00546   ADDBC   LD      A,B             ;routine give vertical increment if needed
00547           OR      A
00548           JR      Z,BC1           ;jump if ball going down
00549           EX      AF,AF'
00550           PUSH    AF
00551           EX      AF,AF'
00552           POP     AF
00553           CP      81H
00554           JR      Z,BC3           ;jump for ball in char. bottom just printed
00555           JR      BC2
00556   BC1     EX      AF,AF'          ;as above, for ball going down
00557           PUSH    AF
00558           EX      AF,AF'
00559           POP     AF
00560           CP      81H
00561           JR      NZ,BC3
00562   BC2     ADD     HL,BC           ;give vertical increment
00563           LD      A,0             ;set A to increment given = 0
00564           RET
00565   BC3     LD      A,0FFH          ;set A to no increment given = -1
00566           RET
```

Following are the routines to bounce the ball when necessary from horizontal, vertical and diagonal obstructions.

```
00567   BNCE01  LD      A,(HL)          ;routine for horizontal bounce (shallow/norm)
00568           CP      0A4H            ;ASCII of screen wall
00569           JR      Z,BNCECT        ;jump if bouncing off wall
00570           EX      AF,AF'
00571           PUSH    AF
00572           EX      AF,AF'
00573           POP     AF
00574           CP      80H             ;if last ball printed was 81H and not
00575           RET     Z               ;       hitting wall, no bounce required
00576           CALL    SCORE           ;adjust score and remove brick
00577   BNCECT  LD      A,E
00578           CP      1
00579           JR      Z,BNCE1B
00580           LD      DE,1            ;if ball had left increment, then give it
00581           RET                     ;       a right increment, and vice
00582   BNCE1B  LD      DE,-1           ;       versa
00583           RET
00584   BNCE02  LD      A,H             ;vertical bounce routine (shallow/norm)
00585           CP      0F0H            ;routine to increase ball speed once per
00586           JR      NZ,BNCE2E       ;       frame as ball hits top of screen
00587           LD      A,L
00588           CP      40H             ;if H = F0H and L <= 40H then the ball is
00589           JR      NC,BNCE2E       ;       bouncing off the screen top
00590           LD      A,0
```

```
00591           LD      (DELAY),A       ;load LSB of DELAY with zero (was FFH)
00592   BNCE2E  LD      A,B             ;if ball moving down then BC = 40H, hence
00593           OR      A               ;       B = 0
00594           JR      NZ,BNCE2D       ;if ball moving up, then jump
00595           LD      A,H             ;this routine will only be called with
00596           CP      0F3H            ;       H = F3H if ball hits screen bottom
00597           JR      NZ,BNCE2C       ;if ball not at screen bottom, then jump
00598           LD      A,(HL)
00599           CP      88H             ;ASCII of screen bottom
00600           JR      NZ,BNCE2F       ;if not 88H, then paddle must be hit
00601           CALL    CORNER          ;check if ball can bounce off corner of paddle
00602           JR      Z,BNCE2G        ;if so, then jump
00603           JP      LSEMAN          ;no conditions met, so a ball is lost
00604   BNCE2F  LD      BC,-40H         ;paddle bounce routine - reverse vert. dirn.
00605           PUSH    AF              ;save A register
00606           LD      A,E
00607           CP      1
00608           JR      NZ,BNCE2J       ;if ball going left, then jump
00609           POP     AF
00610           CP      86H
00611           JP      Z,BO5RET        ;if righthand end of paddle hit (86H
00612           CP      87H             ;       or 87H) then go to steep
00613           JP      Z,BO5RET        ;       bounce routine
00614           CP      82H
00615           JR      Z,BNCE2I        ;if lefthand end of paddle hit (82H
00616           CP      83H             ;       or 83H) then go to shallow
00617           JR      Z,BNCE2I        ;       bounce routine
00618           JP      BNCE2K          ;middle must be hit - normal bounce
00619   BNCE2J  POP     AF              ;same as above routine - selects
00620           CP      82H             ;       bounce type according to
00621           JP      Z,BO5RET        ;       area of paddle hit, except
00622           CP      83H             ;       ball is moving left, and so
00623           JP      Z,BO5RET        ;       bounce types are vice versa
00624           CP      86H
00625           JR      Z,BNCE2I
00626           CP      87H
00627           JR      Z,BNCE2I
00628           JP      BNCE2K
00629   BNCE2I  LD      A,1
00630           LD      (DIRTYP),A      ;load direction type with 1 = shallow bounce
00631           JP      BO2RET
00632   BNCE2K  LD      A,0
00633           LD      (DIRTYP),A      ;load direction type with 0 = normal bounce
00634           LD      A,1
00635           LD      (BETNEC),A      ;load ball between print with 1 = not needed
00636           JP      BO2RET
00637   BNCE2G  LD      BC,-40H         ;bounce off paddle corner - vert. dirn.
00638           LD      A,E             ;       reversed and then horiz. dirn.
00639           CP      1
00640           JR      Z,BNCE2H
00641           LD      DE,1
00642           JP      BO2RET
00643   BNCE2H  LD      DE,-1
00644           JP      BO2RET
00645   BNCE2D  CALL    SCORE           ;ball moving up has only one vertical bounce
```

```
00646           LD      BC,40H          ;       type - adjust score, reverse dirn.
00647           JP      B02RET          ;       and return
00648   BNCE2C  ADD     HL,DE           ;routine to simulate ball bouncing on top
00649           LD      A,(HL)          ;       of a brick
00650           CP      20H
00651           JP      Z,B02RET
00652           POP     HL              ;operates similarly to LOOP07 and EMSC02
00653           PUSH    HL
00654           ADD     HL,DE
00655           ADD     HL,BC
00656           PUSH    HL
00657           CALL    SCORE
00658           POP     HL
00659           EXX
00660           POP     HL
00661           LD      (HL),20H
00662           EXX
00663           LD      (HL),81H
00664           CALL    PADLMV
00665           LD      BC,-40H
00666           LD      A,80H
00667           EX      AF,AF'
00668           LD      A,(BRICKS)
00669           CP      5BH
00670           JP      Z,WSTART
00671           LD      A,1
00672           LD      (BETNEC),A
00673           JP      EMSC01          ;end of top of brick bounce
00674   BNCE03  CALL    SCORE           ;routine to bounce off a brick corner
00675           LD      A,E
00676           CP      1
00677           JR      Z,BNCE3B        ;adjust horizontal increment
00678           LD      DE,1
00679           JR      BNCE3C
00680   BNCE3B  LD      DE,-1
00681   BNCE3C  LD      A,C
00682           CP      40H
00683           JR      Z,BNCE3D
00684           LD      BC,40H
00685           RET                     ;adjust vertical increment
00686   BNCE3D  LD      BC,-40H
00687           RET
00688   BNCE04  CP      0A4H            ;horizontal bounce in steep mode - see
00689           JR      Z,BNCE4B        ;       BNCE01
00690           LD      A,B
00691           OR      A
00692           JP      NZ,B05RET
00693   BNCE4B  LD      A,E
00694           CP      1
00695           JR      Z,BNCE4C
00696           LD      DE,1
00697           JR      BNCE4D
00698   BNCE4C  LD      DE,-1
00699   BNCE4D  LD      A,(HL)
00700           CP      0A4H
```

```
00701           JP      Z,B04RET
00702           LD      BC,-40H
00703           CALL    SCORE
00704           JP      B04RET          ;end horizontal steep bounce
00705   BNCE05  LD      A,H             ;vertical bounce in steep mode - see
00706           CP      0F0H            ;       BNCE02
00707           JR      NZ,BNCE5C
00708           LD      A,L
00709           CP      40H
00710           JR      NC,BNCE5C
00711           LD      A,0
00712           LD      (DELAY),A
00713   BNCE5C  LD      A,B
00714           OR      A
00715           JR      Z,BNCE5B
00716           CALL    SCORE
00717           LD      BC,40H
00718           JP      B5CRET
00719   BNCE5B  LD      A,H
00720           CP      0F3H
00721           JR      Z,BNCE5D
00722           ADD     HL,DE
00723           LD      A,(HL)
00724           CP      20H
00725           JP      Z,B5CRET
00726           POP     HL
00727           LD      (HL),20H
00728           ADD     HL,DE
00729           ADD     HL,BC
00730           PUSH    HL
00731           CALL    SCORE
00732           POP     HL
00733           LD      (HL),81H
00734           CALL    PDLMV2
00735           LD      A,(BRICKS)
00736           CP      5BH
00737           JP      Z,WSTART
00738           LD      BC,-40H
00739           JP      EMSC03
00740   BNCE5D  ADD     HL,DE
00741           LD      A,(HL)
00742           CP      88H
00743           JP      Z,LSEMAN
00744           LD      A,(HL)
00745           POP     HL
00746           LD      (HL),20H
00747           ADD     HL,DE
00748           PUSH    HL
00749           LD      (HL),80H
00750           PUSH    AF
00751           CALL    PDLMV2
00752           POP     AF
00753           JP      BNCE2F          ;end vertical steep bounce
00754   CORNER  ADD     HL,DE           ;routine to check for ball bouncing off
00755           LD      A,(HL)          ;       corner of paddle - zero flag set if
```

```
00756           SBC     HL,DE           ;       next ball position is a paddle corner
00757           CP      82H             ;left hand paddle corner
00758           JR      NZ,CRNR02
00759           RET
00760   CRNR02  CP      87H             ;right hand paddle corner
00761           RET
```

The score adjusting routines begin here.

The first routine determines which part of a brick, if any, the ball has been bounced off. If a brick has been hit, the score is adjusted accordingly and the brick removed from the screen.

```
00762   SCORE   LD      A,(HL)          ;brick removing and score adjusting routine
00763           CP      94H
00764           JR      Z,BRICK1
00765           CP      95H             ;select part of brick hit
00766           JR      Z,BRICK2
00767           CP      96H
00768           JR      Z,BRICK3
00769           RET                     ;return if not brick hit (eg screen top)
00770   BRICK1  CALL    SCRADJ          ;adjust score
00771           LD      (HL),20H
00772           INC     HL
00773           LD      (HL),20H        ;blank out three brick sections
00774           INC     HL
00775           LD      (HL),20H
00776           RET
00777   BRICK2  CALL    SCRADJ          ;as for BRICK1
00778           LD      (HL),20H
00779           INC     HL
00780           LD      (HL),20H
00781           DEC     HL
00782           DEC     HL
00783           LD      (HL),20H
00784           RET
00785   BRICK3  CALL    SCRADJ          ;as for BRICK1
00786           LD      (HL),20H
00787           DEC     HL
00788           LD      (HL),20H
00789           DEC     HL
00790           LD      (HL),20H
00791           RET
```

This routine determines what score is given for a particular brick according to how far back in the wall it is. This score is then added onto the player's score. The no. of bricks hit so far is also incremented.

```
00792   SCRADJ  PUSH    DE              ;routine to adjust score
00793           LD      A,(SNDOFF)
00794           CP      FALSE
00795           JR      NZ,CONT4        ;jump if sound not needed
00796           PUSH    BC
00797           LD      C,30H           ;tone length
```

```
00798           LD      D,128           ;tone on
00799           LD      E,128           ;tone off
00800           CALL    NOTE
00801           POP     BC
00802   CONT4   LD      A,H             ;select score to be given according
00803           CP      0F0H            ;       to brick position on
00804           JR      NZ,ADJ02        ;       screen, and load it into DE
00805           LD      DE,16
00806           LD      A,L
00807           CP      0BFH
00808           JR      C,SCRCTD
00809           LD      DE,5
00810           JR      SCRCTD
00811   ADJ02   LD      DE,4
00812           LD      A,L
00813           CP      3FH
00814           JR      C,SCRCTD
00815           LD      DE,3
00816           CP      7FH
00817           JR      C,SCRCTD
00818           LD      DE,2
00819           CP      0BFH
00820           JR      C,SCRCTD
00821           LD      DE,1
00822   SCRCTD  LD      (SCORE1),DE
00823           POP     DE
00824   INITSC  PUSH    HL
00825           PUSH    DE
00826           LD      A,(SCORE1)
00827           LD      HL,SCORE2
00828           ADD     A,(HL)          ;add points just scored to old score
00829           DAA                     ;       and decimal adjust
00830           LD      (SCORE2),A
00831           LD      A,(SCOR1B)
00832           INC     HL
00833           ADC     A,(HL)
00834           DAA                     ;adjust for any tens increment
00835           LD      (SCOR2B),A
00836           PUSH    BC
00837           LD      HL,SCORE2
00838           LD      BC,SCORPS
```

IF YOU ARE USING 64K MICROBEE WITH DISK DRIVE, THE NEXT LINE SHOULD READ:

```
00839           LD      D,97H
```

ELSE THE NEXT LINE SHOULD READ:

```
00839           LD      D,30H
```

CONTINUE AS NORMAL.....

```
00840           CALL    PRSCOR          ;print the new score
00841           POP     BC
```

```
00842           LD      A,(BRICKS)
00843           INC     A               ;increment no. of bricks hit
00844           LD      (BRICKS),A
00845           CALL    EXTMAN          ;check if extra ball should be awarded
00846           POP     DE
00847           POP     HL
00848           RET
```

This routine prints a score from (HL) to the screen position (BC).

```
00849   PRSCOR  LD      A,0             ;routine to print a score at (BC)
00850           RRD
00851           PUSH    AF
00852           ADD     A,D             ;ones figure printed from the least significant
00853           LD      (BC),A          ;       nybble of the score LSB
00854           POP     AF
00855           RRD
00856           PUSH    AF
00857           ADD     A,D
00858           DEC     BC
00859           LD      (BC),A          ;tens from MSN of LSB
00860           POP     AF
00861           RRD
00862           INC     HL
00863           RRD
00864           PUSH    AF
00865           ADD     A,D
00866           DEC     BC
00867           LD      (BC),A          ;hundreds from LSN of MSB
00868           POP     AF
00869           RRD
00870           PUSH    AF
00871           ADD     A,D
00872           DEC     BC
00873           LD      (BC),A          ;thousands from MSN of MSB
00874           POP     AF
00875           RRD
00876           RET
```

An extra man is awarded here if 400 points has been reached. The current score is checked for being over 400: if it is, then LSTHUN is checked to ensure that 400 had not been reached last time EXTMAN was called. If this condition is met, the no. of balls remaining is incremented.

```
00877   EXTMAN  LD      A,(SCOR2B)      ;routine to determine if extra ball awarded
00878           CP      4               ;check for four hundreds reached
00879           RET     NZ
00880           LD      A,(LSTHUN)      ;if reached, check for not reached after
00881           CP      4               ;       last points scored
00882           LD      A,5
00883           LD      (LSTHUN),A
00884           RET     NC              ;return if it was already reached
00885           LD      A,(MEN)
00886           INC     A               ;increment balls remaining
```

```
00887           LD      (MEN),A         ;       and print it
```

IF YOU ARE USING 64K MICROBEE WITH DISK DRIVE, THE NEXT LINE SHOULD READ:

```
00888           ADD     A,97H
```

ELSE THE NEXT LINE SHOULD READ:

```
00888           ADD     A,30H
```

CONTINUE AS NORMAL.....

```
00889           LD      (MENPOS),A
00890           LD      A,(SNDOFF)
00891           OR      A
00892           JR      NZ,CONT5
00893           PUSH    HL              ;routine to play "extra ball" sound
00894           PUSH    BC
00895           LD      HL,0
00896   MNLOP1  LD      B,50
00897           LD      A,SPKRON
00898           OUT     (2),A
00899   MNLOP2  DJNZ    MNLOP2
00900           LD      B,50
00901           LD      A,SPKROF
00902           OUT     (2),A
00903   MNLOP3  DJNZ    MNLOP3
00904           LD      A,0
00905   MNLOP4  INC     A
00906           CP      L
00907           JR      NZ,MNLOP4
00908           DEC     L
00909           JR      NZ,MNLOP1
00910           INC     H
00911           LD      A,H
00912           CP      3
00913           JR      NZ,MNLOP1
00914           POP     BC
00915           POP     HL
00916   CONT5   RET
```

If a ball hits the bottom of the screen, it is 'out' and the player loses a ball. This subroutine performs this, and if the player has lost all his balls, it jumps to GAMEND.

```
00917   LSEMAN  CALL    WAIT            ;routine to print ball touching bottom of
00918           POP     HL              ;       screen and restart frame or game
00919           LD      (HL),20H
00920           ADD     HL,DE
00921           ADD     HL,BC
00922           LD      (HL),0A1H
00923           LD      A,(SNDOFF)
00924           OR      A
00925           JR      NZ,CONT3
```

```
00926           PUSH    HL
00927           LD      C,128           ;play "lose man" sound (see page 109)
00928           LD      B,255
00929           LD      L,0             ;for 2 MHz Bee type LD  L,170
00930   LSLOP1  LD      E,C
00931           LD      D,B
00932           LD      A,SPKRON
00933           OUT     (2),A
00934   LSLOP2  DEC     D
00935           JR      NZ,LSLOP2
00936           LD      A,SPKROF
00937           OUT     (2),A
00938   LSLOP3  DEC     E
00939           JR      NZ,LSLOP3
00940           LD      A,0
00941   LSLOP4  INC     A
00942           CP      L
00943           JR      NZ,LSLOP4
00944           INC     L
00945           JR      NZ,LSLOP1
00946           POP     HL
00947   CONT3   LD      A,(MEN)
00948           DEC     A               ;decrease no. of men left
00949           LD      (MEN),A
```

IF YOU ARE USING 64K MICROBEE WITH DISK DRIVE, THE NEXT LINE SHOULD READ:

```
00950           ADD     A,97H
```

ELSE THE NEXT LINE SHOULD READ:

```
00950           ADD     A,30H
```

CONTINUE AS NORMAL.....

```
00951           LD      (MENPOS),A      ;and print it
00952           LD      BC,0FFFFH
00953   LSLOOP  DEC     BC
00954           LD      A,B
00955           OR      C
00956           JR      NZ,LSLOOP       ;delay when ball lost
00957           LD      A,(MEN)
00958           OR      A
00959           JR      Z,GAMEND        ;check for game over
00960           LD      (HL),88H        ;reset ball lost position
00961           JP      LOOP09          ;restart with new ball
```

The routine to enter a new high score if necessary is called from here. The high score and high scoring player's name is then displayed, along with the current player's score and a comment on his ability, according to his score.

```
00962   GAMEND  CALL    CLS             ;game finished routine
00963           CALL    NEWHI           ;set new high score if necessary
```

```
00964           LD      HL,SCRN12
00965           LD      DE,0F14BH
00966           LD      BC,17
00967           LDIR                    ;print "high score is" message
00968           LD      HL,SCORE3
00969           LD      BC,0F160H
00970           LD      D,30H
00971           CALL    PRSCOR          ;print high score
00972           LD      HL,SCRN13
00973           LD      DE,0F161H
00974           LD      BC,8
00975           LDIR                    ;print "set by" message
00976           LD      HL,HISCOR
00977           LD      DE,0F16AH
00978           LD      BC,10
00979           LDIR                    ;print name of high score holder
00980           LD      HL,SCRN10
00981           LD      DE,0F1D6H
00982           LD      BC,14
00983           LDIR                    ;print "your score was" message
00984           LD      HL,SCORE2
00985           LD      BC,0F1E8H
00986           LD      D,30H
00987           CALL    PRSCOR          ;print player's score
00988           LD      HL,SCRN0C       ;begin selecting standard of play message
00989           LD      DE,0F216H       ;       according to score
00990           LD      BC,19
00991           LD      A,(SCOR2B)
00992           CP      1               ;score between 0 - 100
00993           JR      C,PRMESG
00994           LD      HL,SCRN0D
00995           LD      DE,0F210H
00996           LD      BC,32
00997           CP      2               ;score between 101 - 200
00998           JR      C,PRMESG
00999           LD      HL,SCRN0E
01000           LD      DE,0F20FH
01001           LD      BC,34
01002           CP      3               ;score between 201 - 300
01003           JR      C,PRMESG
01004           LD      HL,SCRN11
01005           LD      DE,0F211H
01006           LD      BC,29
01007           CP      4               ;score between 301 - 400
01008           JR      C,PRMESG
01009           LD      HL,SCRN0F       ;score must now be >400
01010           LD      DE,0F20FH
01011           LD      BC,34           ;end selecting standard of play message
01012   PRMESG  LDIR                    ;print standard of play message
01013           LD      H,8H
01014   SCLOP1  LD      BC,0FFFFH       ;delay whilst end of game messages
01015   SCLOP2  DEC     BC              ;       displayed on screen
01016           LD      A,B
01017           OR      C
01018           JR      NZ,SCLOP2
```

```
01019           DEC     H
01020           LD      A,H
01021           OR      A
01022           JR      NZ,SCLOP1
01023           JP      NEWGME          ;restart the game
```

This routine determines if a new high score has been reached, and if so, makes the current player's score the new high score and allows him to enter his name as high scorer's name. A range of letters 'A' - 'Z', '.' and ' ' are allowed to be entered. Incorrect letters may be backspaced over, and when all the desired letters are entered (10 maximum) <RETURN> must be pressed.

```
01024   NEWHI   LD      HL,(SCORE3)     ;routine to determine if a new high score has
01025           LD      A,(SCOR2B)      ;       been attained, and enter a name if so
01026           CP      H               ;return if MSB of high score is higher than
01027           RET     C               ;       the MSB of the new score
01028           JR      NZ,HINAME       ;       and jump if less
01029           LD      A,(SCORE2)      ;return if MSB's are equal and the high
01030           CP      L               ;       score LSB is greater than the
01031           RET     C               ;       new score LSB
01032   HINAME  LD      A,(SCORE2)
01033           LD      (SCORE3),A
01034           LD      A,(SCOR2B)
01035           LD      (SCOR3B),A      ;load high score memory with the new score
01036           LD      HL,SCRN14
01037           LD      DE,0F189H
01038           LD      BC,45
01039           LDIR                    ;print "congratulations" message
01040           LD      HL,SCRN15
01041           LD      DE,0F1D2H
01042           LD      BC,17
01043           LDIR                    ;print "enter your name" message
01044           LD      E,250
01045           LD      HL,CGTDAT
01046           LD      A,(SNDOFF)
01047           OR      A
01048           CALL    Z,PLYTUN        ;play "new high score" tune
01049           LD      HL,0F1E4H       ;lefthand end of high score name on screen
01050           LD      DE,HISCOR       ;lefthand end of high score name in memory
01051           LD      B,10            ;length of high score name
01052   NMLOOP  LD      A,2DH           ;keyboard code of character before "." -
01053   LTLOOP  INC     A               ;       bottom of keyboard search
01054           CP      5BH             ;keyboard code of character after "Z" -
01055           JR      Z,NMLOOP        ;       top of keyboard search
01056           CALL    TESTKY          ;test for character
01057           JR      NZ,LTLOOP       ;if not found, increment A and try again
01058           CP      34H             ;code for <return>
01059           JP      Z,RETURN
01060           CP      2EH             ;code for "."
01061           JR      Z,FLLSTP
01062           CP      37H             ;code for " "
01063           JR      NZ,NMCONT
01064           SBC     A,17H           ;adjust to ASCII for " "
01065           JR      FLLSTP
```

```
01066  NMCONT  CP    31H          ;code for <backspace>
01067          JR    NZ,NBS
01068          LD    C,A
01069          LD    A,L
01070          CP    0E4H
01071          LD    A,C
01072          JR    Z,LTLOOP     ;if no backspace possible, inc. A and try again
01073          DEC   HL
01074          DEC   DE
01075          INC   B            ;blank entered characters if possible
01076          LD    A,20H
01077          LD    (HL),A
01078          LD    (DE),A
01079          INC   B
01080          JR    BSDEL
01081  NBS     CP    41H
01082          JR    C,LTLOOP
01083  FLLSTP  LD    (HL),A       ;load screen and memory with the last
01084          LD    (DE),A       ;       character found
01085          INC   HL           ;       and adjust the counters
01086          INC   DE
01087  BSDEL   PUSH  AF
01088          PUSH  BC
01089          LD    BC,3800H     ;key debounce delay (adjust value if required)
01090  LETDEL  DEC   BC
01091          LD    A,B
01092          OR    C
01093          JR    NZ,LETDEL
01094          POP   BC
01095          POP   AF
01096          DJNZ  LTLOOP       ;return for next character if not full
01097  RETLP   LD    A,34H        ;if full, wait for a <return>
01098          CALL  TESTKY
01099          JR    Z,RETURN
01100          LD    A,31H        ;       or a <backspace>
01101          CALL  TESTKY
01102          JR    Z,NMLOOP
01103          JR    RETLP
01104  RETURN  LD    A,B          ;if player has entered all the letters
01105          OR    A            ;       wanted, then the rest of the
01106          JR    Z,RET1       ;       high score name must be blanked
01107          LD    A,20H        ;       if necessary
01108  RET2    LD    (HL),A
01109          LD    (DE),A
01110          INC   HL
01111          INC   DE
01112          DJNZ  RET2
01113  RET1    CALL  CLS
01114          RET
```

This routine responds to either joystick input or key input ('<' and '>'). Joystick input is first searched for, and failing this, a key input is searched for. If some input is found, the paddle is moved one character space in the required direction.

The depression of the <ESC> key is also tested here to return the game to the title screen.

```
01115   PADLMV  LD      A,(DIRTYP)      ;routine to move paddle correct no. of times
01116           OR      A               ;check for direction type
01117           JR      NZ,PDLMV2
01118           CALL    PADDLE
01119   PDLMV2  CALL    PADDLE          ;moves paddle less per ball movement to
01120           CALL    PADDLE          ;       adjust for ball speed
01121           CALL    PADDLE
01122           RET
01123   PADDLE  CALL    WAIT            ;routine to move paddle
01124           EXX
01125           PUSH    AF
01126           LD      A,48            ;code of "escape" key
01127           CALL    TESTKY          ;check for escape pressed
01128           JP      Z,ASTART        ;       and start again if so
01129           IN      A,(0)
01130           OR      A               ;check for joystick input
01131           JR      Z,NOJYST        ;jump if none
01132           BIT     3,A             ;check for joystick right
01133           JR      Z,PDLRGT
01134           BIT     2,A             ;check for joystick left
01135           JR      Z,PDLLFT
01136   NOJYST  LD      A,46
01137           CALL    TESTKY          ;check for right arrow pressed
01138           JR      Z,PDLRGT
01139           LD      A,44
01140           CALL    TESTKY          ;check for left arrow pressed
01141           JR      Z,PDLLFT
01142           POP     AF
01143           EXX
01144           RET
01145   PDLRGT  LD      A,E             ;routine to move paddle right one space
01146           CP      0F8H
01147           JR      Z,TOOFAR        ;check for far right border reached
01148           LD      A,88H           ;move lefthand end of paddle right one
01149           LD      (DE),A          ;       space and print the paddle here
01150           INC     DE
01151           LD      HL,SCRN03
01152           PUSH    DE
01153           LD      BC,6
01154           LDIR
01155           POP     DE
01156           POP     AF
01157           EXX
01158           RET
01159   PDLLFT  LD      A,E             ;as for PDLRGT but for left direction
01160           CP      0C1H
01161           JR      Z,TOOFAR
01162           LD      HL,5
01163           ADD     HL,DE
01164           LD      (HL),88H
01165           DEC     DE
01166           LD      HL,SCRN03
```

```
01167           PUSH    DE
01168           LD      BC,6
01169           LDIR
01170           POP     DE
01171   TOOFAR  POP     AF              ;if paddle reaches either edge of
01172           EXX                     ;       screen, then return
01173           RET
```

This subroutine is called each time the paddle movement routine is called to create a delay and slow the ball motion to reasonable levels. The size of DELAY determines the ball speed.

```
01174   WAIT    PUSH    BC              ;delay loop to slow play
01175           LD      BC,(DELAY)
01176   LOOP08  DEC     BC
01177           LD      A,B
01178           OR      C
01179           JR      NZ,LOOP08
01180           POP     BC
01181           RET
```

This routine plays a sequence of notes from (HL).

Sound is produced on the MicroBee by first sending an activating output to the speaker port (port 2), followed by a delay. A deactivating output is then sent to port 2, followed by a second delay. By varying the lengths of the two delays, various tones may be produced, and the whole process may be repeated many times to give the note length.

```
01182   PLYTUN  LD      C,(HL)          ;routine to play a sequence of notes
01183           LD      A,(HL)          ;load note length
01184           OR      A
01185           RET     Z               ;return if a zero length is reached
01186           INC     HL
01187           LD      D,(HL)          ;load time speaker port "high"
01188           CALL    NOTE            ;play the note
01189           INC     HL
01190           JR      PLYTUN          ;return for next note
01191   NOTE    LD      B,D             ;routine to play a note - load time "high"
01192           LD      A,SPKRON        ;load active output into A
01193           OUT     (2),A           ;       and output through speaker port
01194   NOTE1   DEC     B
01195           JR      NZ,NOTE1        ;delay to give pitch
01196           LD      A,SPKROF        ;load inactive output into A
01197           OUT     (2),A           ;       and output through speaker port
01198           LD      B,E
01199   NOTE2   DEC     B               ;pitch delay
01200           JR      NZ,NOTE2
01201           DEC     C
01202           JR      NZ,NOTE         ;continue until length reached
01203           RET
```

TESTKY is used to determine if a certain key whose keyboard code is in the A register has been pressed. If the key is pressed,

then the zero flag is set.

```
01204   TESTKY  PUSH    BC
01205           LD      C,A             ;save value of key being tested
01206           LD      B,A
01207           LD      A,12H
01208           OUT     (0CH),A         ;select update register (high) of 6545
01209           LD      A,B
01210           RRCA
01211           RRCA
01212           RRCA
01213           RRCA
01214           AND     3H
01215           OUT     (0DH),A         ;set high address of key
01216           LD      A,13H
01217           OUT     (0CH),A         ;select update register (low) of 6545
01218           LD      A,B
01219           RLCA
01220           RLCA
01221           RLCA
01222           RLCA
01223           OUT     (0DH),A         ;set low address of key
01224           LD      A,1H
01225           OUT     (0BH),A         ;select ROM read latch on
01226           LD      A,10H
01227           OUT     (0CH),A
01228           IN      A,(0DH)
01229           LD      A,1FH
01230           OUT     (0CH),A
01231           OUT     (0DH),A
01232   WAIT1   IN      A,(0CH)
01233           BIT     7,A
01234           JR      Z,WAIT1
01235           IN      A,(0CH)
01236           CPL
01237           BIT     6,A
01238           LD      A,0
01239           OUT     (0BH),A         ;select ROM read latch off
01240           LD      A,C             ;reset A with original key code
01241           POP     BC
01242           RET
```

A random number from Ø to 61 is generated here, for the initial ball position at the start of a frame and its initial diagonal direction. The randomizing factor is the length of time it takes the player to press the continuation key at the title screen.

```
01243   RANDOM  LD      A,(RANNUM)      ;routine to generate random number (0 - 61)
01244           RRCA                    ;rotate previous random number right one bit
01245           LD      (RANNUM),A      ;       and store
01246           AND     3FH             ;mask bits six and seven
01247           CP      61              ;check less than 61
01248           JR      C,OKAY
01249           SUB     3               ;reduce to <= 61 if not
01250   OKAY    RET
```

This routine fills the screen RAM with blanks (20H).

```
01251  CLS     LD      HL,VDU          ;routine to clear screen
01252          LD      DE,0F001H
01253          LD      BC,3FFH
01254          LD      (HL),20H        ;ASCII character for SPACE
01255          LDIR
01256          RET
```

The remainder of the program is devoted to data for the 6545 driver, the PCG characters, the tunes played, and also the necessary reserved memory space.

```
01257
01258  ;-------------------------------------------------------------------------
01259  ;       6545 (VIDEO DRIVER CHIP) DATA  (data to set 6545 to 64*16 screen format)
01260  ;-------------------------------------------------------------------------
01261          DB      6BH             ;horizontal sync timing constant - use 5FH for a 2MHz Bee
01262          DB      40H             ;number of displayed characters per line (= 64)
01263          DB      51H             ;horizontal sync position - use 4BH for a 2MHz Bee
01264          DB      37H             ;horizontal and vertical sync width
01265          DB      12H             ;vertical screen width
01266          DB      09H             ;vertical sync timing constant
01267          DB      10H             ;number of displayed rows per screen
01268          DB      11H             ;vertical sync position - use 12H for a 2MHz Bee
01269          DB      48H             ;mode control constant
01270          DB      0FH             ;number of scan lines per character
01271          DB      2FH             ;cursor mode and start line (2F = cursor off)
01272          DB      0FH             ;cursor end line
01273          DB      0               ;display start address relative to F000H (high byte)
01274          DB      0               ;display start address relative to F000H (low byte)
01275          DB      0               ;cursor position (high byte)
01276  D6545   DB      0               ;cursor position (low byte)
01277
```

NOTE: EDASM requires all DEFINE BYTE (DB) statements to be on separate lines. The data below has been printed with up to 16 data values under one DB to save printing space, and each data value should be entered in EDASM on a new line with a new DB.

```
01278  ;-------------------------------------------------------------------
01279  ;                       PCG CHARACTER DATA
01280  ;-------------------------------------------------------------------
01281  PCGDAT  DB      0,0,0,0,0,0,0,0,60,126,255,255,255,255,126,60          ;ball at bottom
01282          DB      60,126,255,255,255,255,126,60,0,0,0,0,0,0,0,0          ;ball at top
01283          DB      192,240,252,255,127,63,15,0,0,0,255,255,255,255,255,255 ;paddle(1)
01284          DB      0,0,0,240,255,255,255,0,0,0,255,255,255,255,255,255    ;paddle(2)
01285          DB      0,0,0,0,128,255,255,0,0,0,255,255,255,255,255,255      ;paddle(3)
01286          DB      0,0,0,0,1,255,255,0,0,0,255,255,255,255,255,255        ;paddle(4)
01287          DB      0,0,0,15,255,255,255,0,0,0,255,255,255,255,255,255     ;paddle(5)
01288          DB      3,15,63,255,254,252,240,0,0,0,255,255,255,255,255,255  ;paddle(6)
01289          DB      0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255            ;underline
```

```
IF YOU ARE NOT USING 64K MICROBEE WITH DISK DRIVE, OMIT FOLLOWING 11 LINES:

01290           DB      60,66,64,64,60,2,2,66,60,0,255,255,255,255,255,255      ;"S"
01291           DB      60,66,64,64,64,64,64,66,60,0,255,255,255,255,255,255    ;"C"
01292           DB      60,66,66,66,66,66,66,66,60,0,255,255,255,255,255,255    ;"O"
01293           DB      124,66,66,66,124,66,66,66,66,0,255,255,255,255,255,255  ;"R"
01294           DB      62,64,64,56,64,64,64,64,62,0,255,255,255,255,255,255    ;"E"
01295           DB      126,64,64,64,120,64,64,64,64,0,255,255,255,255,255,255  ;"F"
01296           DB      60,66,66,66,126,66,66,66,66,0,255,255,255,255,255,255   ;"A"
01297           DB      66,102,90,66,66,66,66,66,66,0,255,255,255,255,255,255   ;"M"
01298           DB      124,66,66,66,124,66,66,66,124,0,255,255,255,255,255,255 ;"B"
01299           DB      64,64,64,64,64,64,64,64,126,0,255,255,255,255,255,255   ;"L"
01300           DB      124,16,16,16,16,16,16,16,16,0,255,255,255,255,255,255   ;"T"

IF YOU ARE NOT USING 64K MICROBEE WITH DISK DRIVE, INSERT THE FOLLOWING TWO LINES:

01290           DB      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
01291           ORG     STRTAD+0AA1H

CONTINUE AS NORMAL .....

01301           DB      0,0,0,0,0,0,0,0,251,251,0,223,223,0,251,251             ;brick(1)
01302           DB      0,0,0,0,0,0,0,0,239,239,0,125,125,0,239,239             ;brick(2)
01303           DB      0,0,0,0,0,0,0,0,190,190,0,246,246,0,190,190             ;brick(3)

IF YOU ARE NOT USING 64K MICROBEE WITH DISK DRIVE, OMIT THE FOLLOWING 10 LINES:

01304           DB      60,70,70,74,74,82,82,98,60,0,255,255,255,255,255,255    ;"0"
01305           DB      16,48,80,16,16,16,16,16,124,0,255,255,255,255,255,255   ;"1"
01306           DB      60,66,2,4,8,16,32,64,126,0,255,255,255,255,255,255      ;"2"
01307           DB      60,66,2,2,28,2,2,66,60,0,255,255,255,255,255,255        ;"3"
01308           DB      64,64,64,72,72,126,8,8,8,0,255,255,255,255,255,255      ;"4"
01309           DB      126,64,64,124,2,2,2,66,60,0,255,255,255,255,255,255     ;"5"
01310           DB      60,66,64,64,124,66,66,66,60,0,255,255,255,255,255,255   ;"6"
01311           DB      126,2,2,4,8,16,32,64,64,0,255,255,255,255,255,255       ;"7"
01312           DB      60,66,66,66,60,66,66,66,60,0,255,255,255,255,255,255    ;"8"
01313           DB      60,66,66,66,62,2,2,66,60,0,255,255,255,255,255,255      ;"9"

IF YOU ARE NOT USING 64K MICROBEE WITH DISK DRIVE, INSERT THE FOLLOWING LINE:

01304           ORG     STRTAD+0B71H

CONTINUE AS NORMAL .............

01314           DB      0,0,60,126,255,255,255,255,126,60,255,255,255,255,255,255       ;ball out

01315   ;                       TITLE   SCREEN   LETTERING

01316           DB      0,0,0,192,224,0,184,188,0,244,246,0,190,191,0,247       ;top right "A"
01317           DB      44,110,0,125,125,0,0,0,0,0,125,0,239,239,0,125          ;top centre "A"
01318           DB      255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255 ;blank
01319           DB      251,251,0,223,223,0,251,251,0,223,223,0,251,251,0,223   ;top "I"
01320           DB      239,239,0,125,125,0,0,0,0,0,125,0,239,239,0,125         ;top centre "B"
01321           DB      128,176,0,244,244,0,190,190,0,246,244,0,184,176,0,192   ;top right "B"
01322           DB      239,111,0,29,29,0,15,7,0,5,5,0,3,1,0,1                  ;bottom centre "R"
```

```
01323           DB      128,128,0,192,224,0,176,176,0,240,244,0,188,188,0,247   ;bottom right "R"
01324           DB      0,0,0,3,11,0,27,59,0,95,95,0,251,251,0,223              ;top left "C"
01325           DB      44,110,0,125,125,0,247,193,0,64,64,0,128,128,0,0        ;top centre "C"
01326           DB      0,0,0,176,184,0,246,247,0,0,0,0,0,0,0,0                 ;top right "C"
01327           DB      251,251,0,223,95,0,123,123,0,31,31,0,3,1,0,0            ;bottom left "C"
01328           DB      128,128,0,0,0,0,192,192,0,65,125,0,239,239,0,24         ;bottom centre "C"
01329           DB      0,0,0,0,0,0,0,0,0,247,244,0,184,176,0,128               ;bottom right "C"
01330           DB      1,3,0,5,5,0,15,15,0,29,29,0,47,47,0,125                 ;top centre "K"
01331           DB      190,190,0,244,244,0,176,176,0,240,240,0,160,128,0,128   ;top right "K"
01332           DB      0,0,0,0,0,0,0,0,192,224,240,240,240,240,224,192         ;right 1/2 ball
01333           DB      0,0,0,0,0,0,0,0,3,7,15,15,15,15,7,3                     ;left 1/2 ball
01334           DB      0,0,0,0,0,0,0,0,0,0,0,0,60,126,255,255                  ;top 1/2 ball
01335           DB      255,255,126,60,0,0,0,0,0,0,0,0,0,0,0,0                  ;bottom 1/2 ball
01336           DB      0,0,0,0,60,126,255,255,255,255,126,60,0,0,0,0           ;ball centre
01337           DB      255,255,126,60,0,0,0,0,251,251,0,223,223,0,251,251      ;1/2 with brick(1)
01338           DB      255,255,126,60,0,0,0,0,190,190,0,246,246,0,190,190      ;1/2 with brick(3)
01339
01340   ;-------------------------------------------------------------------------------
01341   ;       MUSIC DATA FOLLOWS: Sequence of DBs is length of note, pitch, length of note etc
01342   ;-------------------------------------------------------------------------------
```

FOR 2 MHZ MICROBEES, IN LINES 1344 TO 1348 INCLUSIVE, REPLACE ALL
OCCURRENCES OF 80 WITH 40, 60 WITH 30, AND 50 WITH 28.

```
01343           DEFR    16D
01344   ROVDAT  DB      50,0FFH,50,0AAH,50,0FFH,60,0AAH,60,55,60,55,60,55,60,55,60
01345           DB      0AAH,60,0AAH,60,0AAH,60,55,60,0AAH,60,55,60,1BH,60,1BH,60,1BH
01346           DB      60,1BH,60,55,60,55,60,55,0
01347   CGTDAT  DB      80,0E1,80,0AAH,80,81,80,68,80,68,80,0E1,80,0E1,80,68,80
01348           DB      81,80,68,80,41,80,41,80,0AAH,80,0AAH,0
01349
01350   ;-----------------------------------------------------------------
01351   ;                       SCREEN MESSAGES
01352   ;-----------------------------------------------------------------
```

IF YOU ARE USING 64K MICROBEE WITH DISK DRIVE, THE FOLLOWING LINES SHOULD READ:

```
01353   SCRN01  DB      0A4,0A4,88,89,8AH,8BH,8CH,8DH,88,97,97,97,97,88,0A4,0A4,0A4
01354           DB      0A4,0A4,0A4,0A4,0A4,0A4,0A4,0A4,88,8EH,8CH,8FH,90,8DH,88,98
01355           DB      88,0A4,0A4,0A4,0A4,0A4,0A4,0A4,0A4,0A4,0A4,0A4,88,91,8FH
01356           DB      92,92,89,88,92,8DH,8EH,93,88,9AH,88,0A4,0A4
```

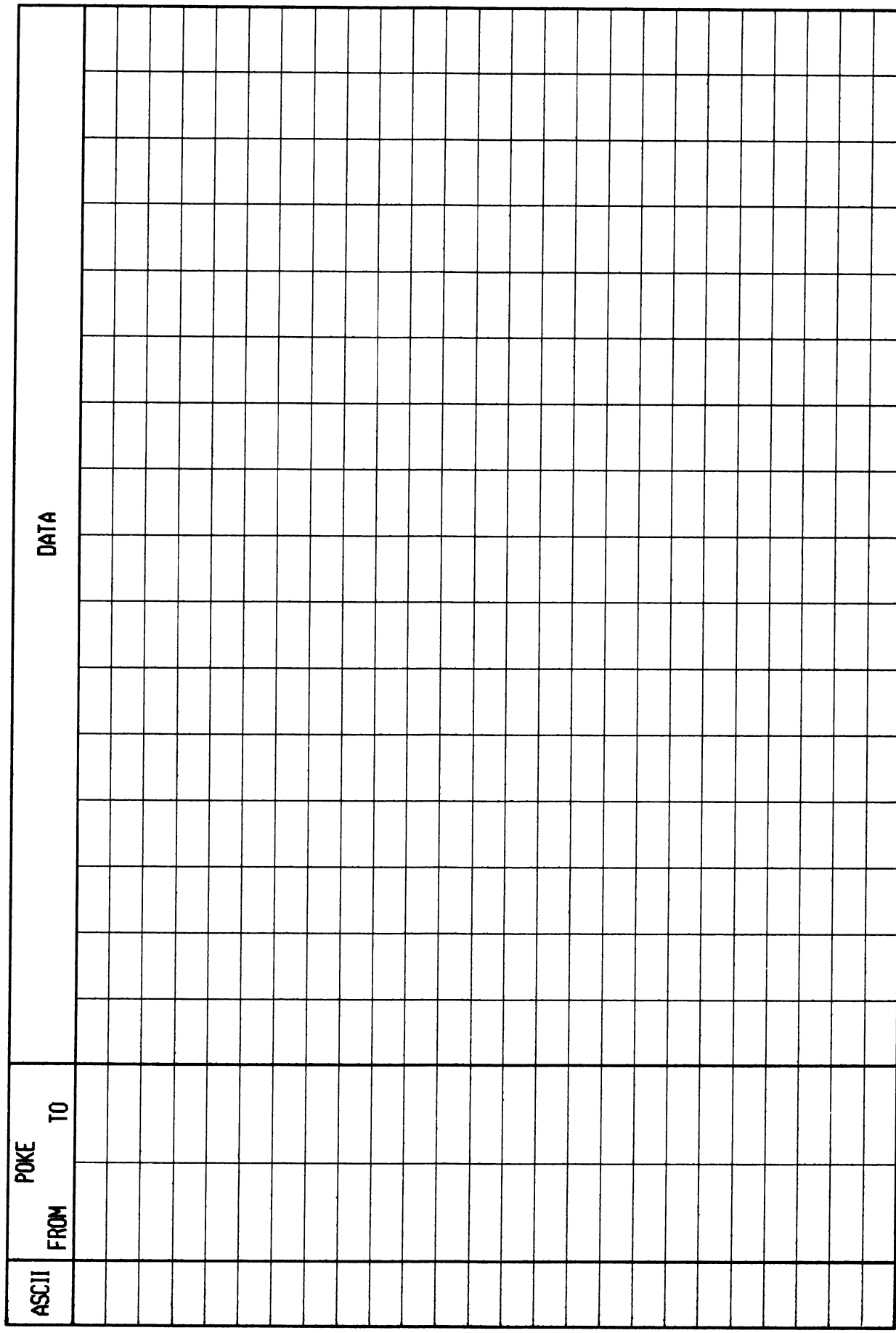
ELSE THEY SHOULD READ:

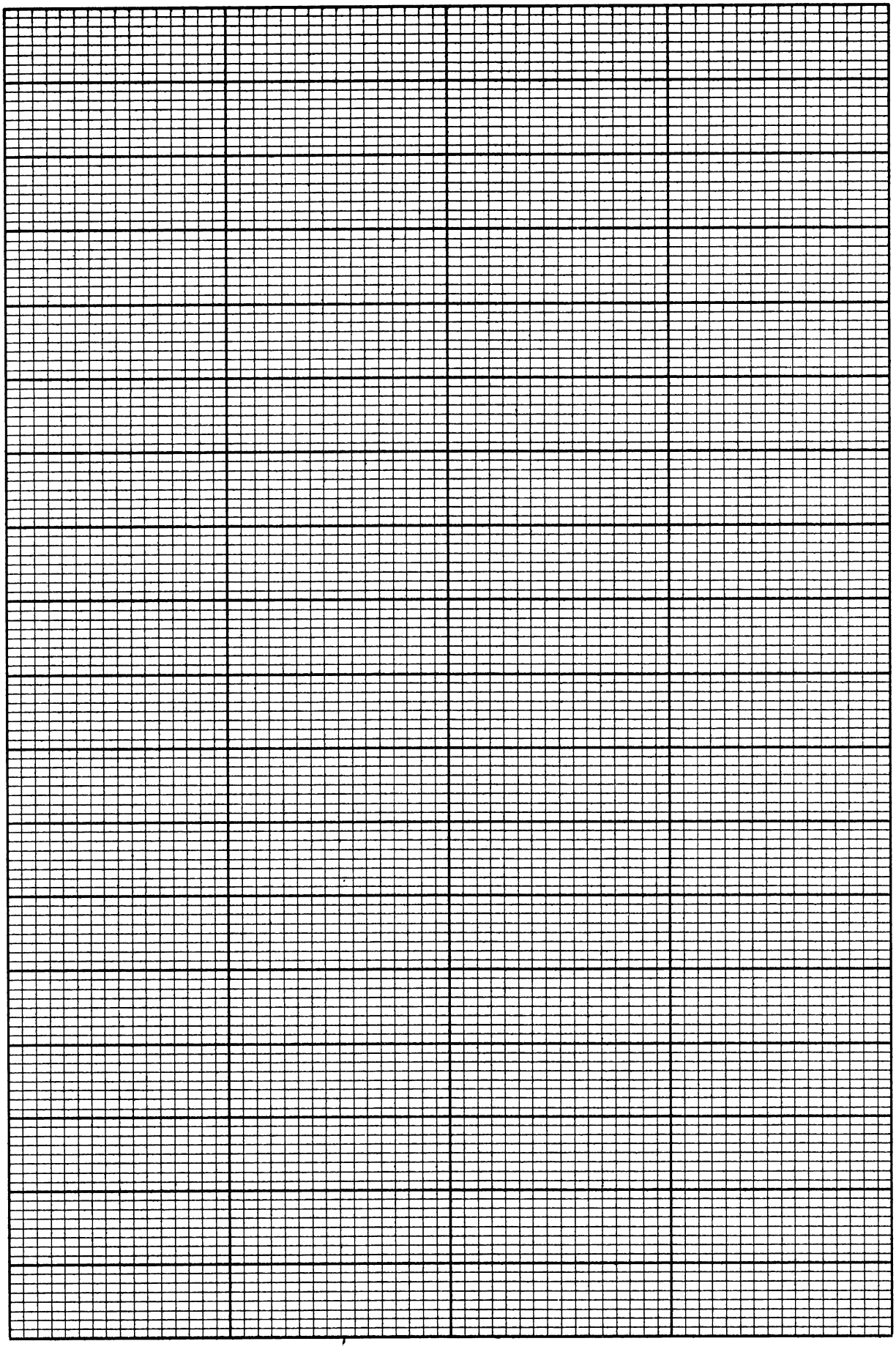
```
01353   SCRN01  DB      0A4,0A4,89,53,43,4FH,52,45,89,30,30,30,30,89,0A4,0A4,0A4
01354           DB      0A4,0A4,0A4,0A4,0A4,0A4,0A4,0A4,89,46,52,41,4DH,45,89,31
01355           DB      89,0A4,0A4,0A4,0A4,0A4,0A4,0A4,0A4,0A4,0A4,0A4,89,42,41
01356           DB      4CH,4CH,53,89,4CH,45,46,54,89,33,89,0A4,0A4
```

CONTINUE AS NORMAL .....

```
01357   BRICK   DB      94,95,96,20
01358   SCRN03  DB      82,83,84,85,86,87
01359   SCRN04  DB      0A5,0A6,0A7,20,0A5,0A6,0A7,20,0A5,20,0AAH,0ABH,0ACH,20,0A5
01360           DB      0B0,0B1,20,94,95,96,20,0A5,0A6,0A7,20,0AAH,0A3,0A2,20,0A5
```

```
01361           DB      20,20,20,0A5,20,20
01362   SCRN05  DB      0A5,0A6,0A7,20,0A5,0A8,0A9,20,0A5,20,0ADH,0AEH,0AFH,20,0A5
01363           DB      0A8,0A9,20,20,20,20,20,0A5,0A6,0A7,20,0A5,20,0A5,20,0A5
01364           DB      95,96,20,0A5,95,96
01365   SCRN06  DEFM    'Written by D.S. Long 1984'
01366   SCRN07  DEFM    'By manipulating the paddle with "<" and ">",'
01367   SCRN08  DEFM    'rebound the ball to knock out bricks.'
01368   SCRN09  DB      0A4,0D0,0CFH,0C9,0CEH,0D4,0D3,0A4,0D3,0C3,0CFH,0D2,0C5,0A4
01369   SCRN0A  DEFM    'ROW 1:1   ROW 2:2   ROW 3:3   ROW 4:4   ROW 5:5   ROW 6:10'
01370   SCRN0B  DEFM    'Press any key to continue.'
01371   SCRN0C  DEFM    'OBVIOUSLY A NOVICE!'
01372   SCRN0D  DEFM    'STILL MUCH ROOM FOR IMPROVEMENT!'
01373   SCRN0E  DEFM    'KEEP TRYING, YOU`RE GETTING THERE!'
01374   SCRN0F  DEFM    'CONGRATULATIONS!  YOU`RE A MASTER!'
01375   SCRN10  DEFM    'YOUR SCORE WAS'
01376   SCRN11  DEFM    'YOU ARE GETTING QUITE EXPERT!'
01377   SCRN12  DEFM    'THE HIGH SCORE IS'
01378   SCRN13  DEFM    ', SET BY'
01379   SCRN14  DEFM    'CONGRATULATIONS!  YOU HAVE THE NEW HIGH SCORE!'
01380   SCRN15  DEFM    'ENTER YOUR NAME :'
01381   SCRN16  DEFM    'SELECT SPEED (1 = FAST, 3 = SLOW)'
01382   SCRN17  DEFM    'FRAME'
01383   SCRN18  DEFM    'DO YOU WANT SOUND (Y/N)?'
01384
01385   ;-------------------------------------------------------------------------
01386   ;                       RESERVED MEMORY SPACE
01387   ;-------------------------------------------------------------------------
01388   SCORE1  DS      1
01389   SCOR1B  DS      1
01390   SCORE2  DS      1
01391   SCOR2B  DS      1
01392   SCORE3  DB      0
01393   SCOR3B  DB      1
01394   FRAME   DS      1
01395   MEN     DS      1
01396   BRICKS  DS      1
01397   LSTHUN  DS      1
01398   DELAY   DS      2
01399   HISCOR  DEFM    'MICROBEE. '
01400   DIRTYP  DS      1
01401   BETNEC  DS      1
01402   SNDOFF  DS      1
01403   RANNUM  DEFS    1
01404           DEFS    80
01405   STKPTR  EQU     $
01406           END
```

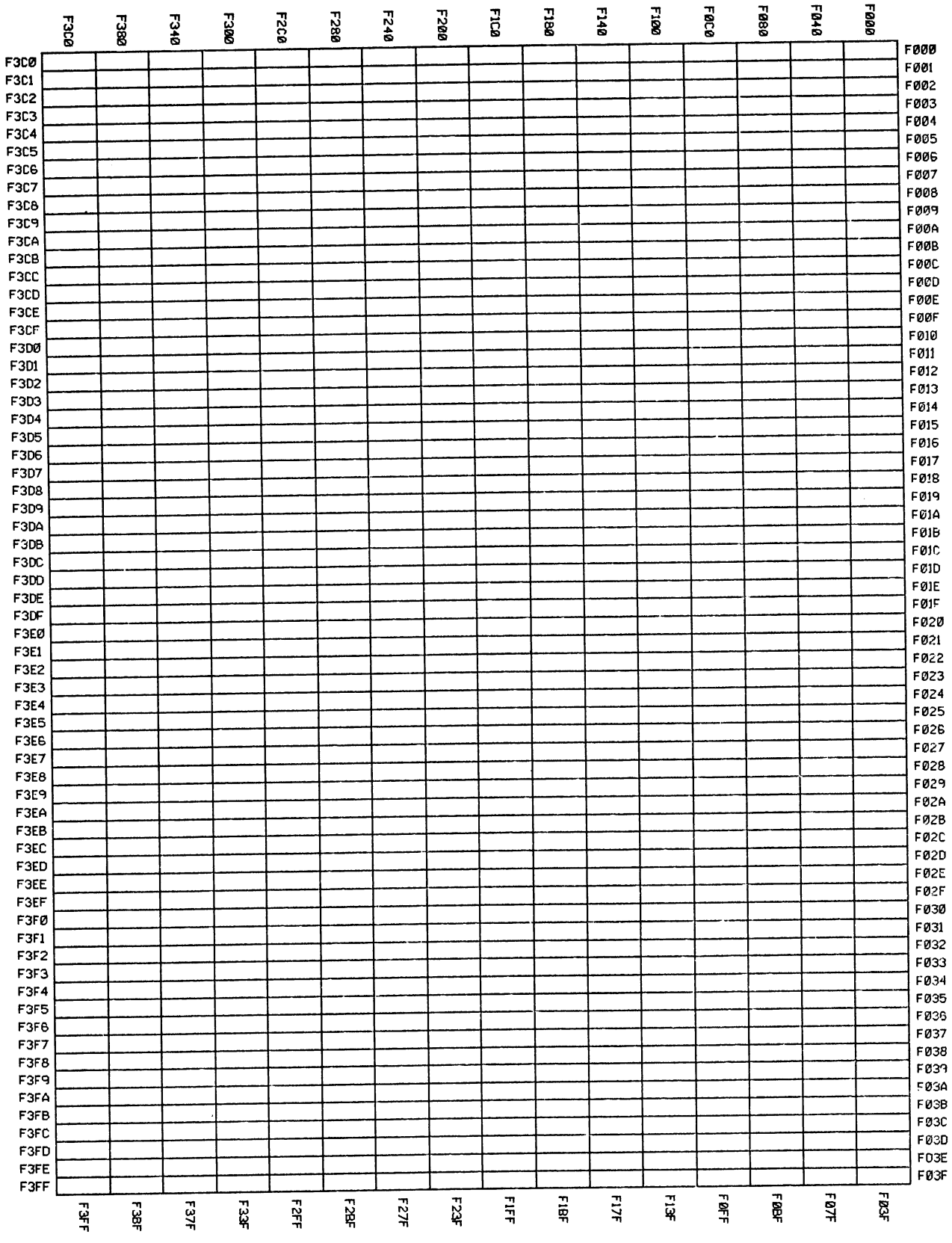

ASCII
POKE FROM
TO
DATA



ASSEMBLER PROGRAMMING SHEET

ADDRESS	LABEL	MNEMONIC	OPERAND	COMMENT

PROGRAM: DATE: SHEET:


F3C0 F380 F340 F300 F2C0 F280 F240 F200 F1C0 F180 F140 F100 F0C0 F080 F040 F000
F3C0 F3C1 F3C2 F3C3 F3C4 F3C5 F3C6 F3C7 F3C8 F3C9 F3CA F3CB F3CC F3CD F3CE F3CF F3D0 F3D1 F3D2 F3D3 F3D4 F3D5 F3D6 F3D7 F3D8 F3D9 F3DA F3DB F3DC F3DD F3DE F3DF F3E0 F3E1 F3E2 F3E3 F3E4 F3E5 F3E6 F3E7 F3E8 F3E9 F3EA F3EB F3EC F3ED F3EE F3EF F3F0 F3F1 F3F2 F3F3 F3F4 F3F5 F3F6 F3F7 F3F8 F3F9 F3FA F3FB F3FC F3FD F3FE F3FF
F000 F001 F002 F003 F004 F005 F006 F007 F008 F009 F00A F00B F00C F00D F00E F00F F010 F011 F012 F013 F014 F015 F016 F017 F018 F019 F01A F01B F01C F01D F01E F01F F020 F021 F022 F023 F024 F025 F026 F027 F028 F029 F02A F02B F02C F02D F02E F02F F030 F031 F032 F033 F034 F035 F036 F037 F038 F039 F03A F03B F03C F03D F03E F03F
F3FF F38F F37F F33F F2FF F2BF F27F F23F F1FF F1BF F17F F13F F0FF F08F F07F F03F

WRITE FOR US

This volume was compiled with the help of many contributors who all received payment for their efforts. If you have written an original program which could be of general appeal, or a subroutine, or an article, please send it to us for possible inclusion in future publications. Anything from a short tip of a few lines to a complete book will be considered. Write first if you have any queries.

We wish to point out that the memory maps published in earlier volumes were for information only, and as new versions of ROM-based programs are released, routines may be relocated, modified, or deleted altogether, and the memory maps become unreliable. If you are writing programs for commercial release, do not make USR calls in BASIC, or their equivalent in assembly language, or else you risk their failure to run in later models of the MicroBee.

To make it easier for us to use your contribution without much editing, we suggest the following guidelines:

Acceptable formats	: WordStar, Word-Bee, EDASM, BASIC
Media	: Tape or Disk
Text structure	: Line length = 66 columns
	: No escape sequences or dot commands
	: Leave spaces after BASIC keywords
	: Two spaces after a full stop; one after a comma
	: Use single quotation marks in text, and do not use opening quote (Shift @)
	: REMs/Source Code comments in lower case
	: Source code to use equates, with limited calls to BASIC/EDASM ROMs
	: No calls to Boot, Net and Telcom ROMs
	: All programs within the text must be supplied on tape or disk
Content	: Well-commented BASIC or Source code
	: Tips and Techniques
	: Tutorial style
	: Colour and Disk topics
	: Games using M/L sound and PCG graphics
	: Cartoons
	: Screen Dumps

Address ONLY for sending contributions:

B F & N Publishing, PO Box 511, Werribee VIC 3030, Australia.

USER GROUPS

To get full value from your hobby, it pays to belong to a club which consists of other people with the same interest. There are many clubs for computer hobbyists, known as User Groups. The ones in the large cities have hundreds of members, meet once or twice a month, and publish a newsletter full of programs, hints and tips. Many have a software library containing non-commercial software for loan to members. Some sell commercial software at a discount, or organise bulk purchase of hardware, printer paper, magnetic media etc. The newsletters depend on contributions from members, so their quality waxes and wanes. Membership costs about A$20.00 per annum for local members, but in all cases write first to the club concerned. The following list is not complete, as we have not included clubs whose existence (in mid 1984) has not been confirmed by sighting a newsletter or by a reply to our letters. If there is no club in your area, you should consider forming at least an informal group that meets in private homes. If your club is not mentioned below, let us know, and we will include the information in the next edition:

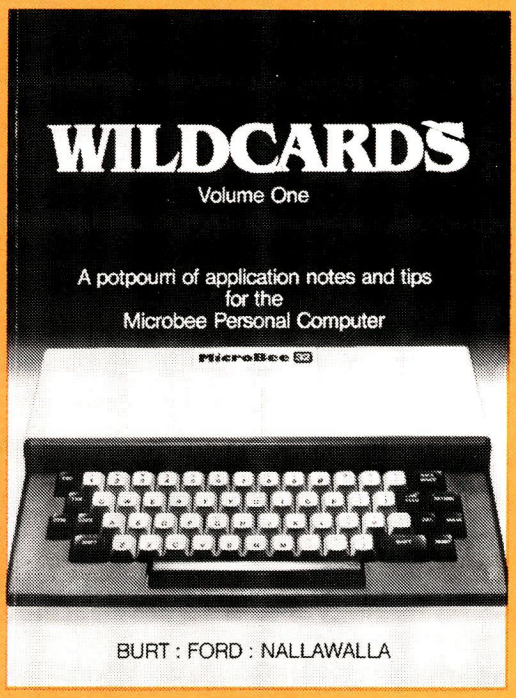
1. BENDIGO: c/- 33 Nolan St., Bendigo VIC 3550
2. BRISBANE: P.O. Box 332, Ashgrove QLD 4060
3. CANBERRA: c/- 6 Boyland Close, Spence ACT 2615
4. DARWIN: G.P.O. Box 5111, Darwin NT 5794
5. MELBOURNE: P.O. Box 157, Nunawading VIC 3131
6. MELBOURNE: P.O. Box 88, St. Albans VIC 3012
7. PERTH: c/- 4 Garnkirk Road, Greenwood WA 6024
8. SYDNEY: P.O. Box C233, Clarence St., Sydney NSW 2000

OVERSEAS:

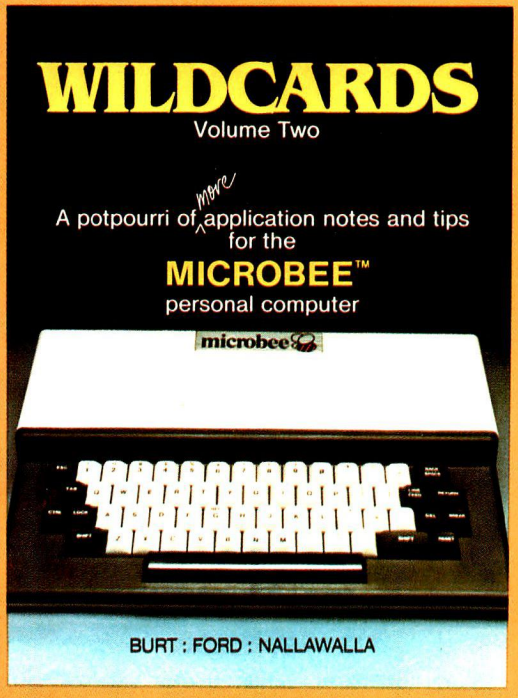
1. STOCKHOLM: c/- Rolf Lindgren, Linguistics Dept, University of Stockholm, S-106 91, SWEDEN

INDEX

OTHER BOOKS IN THE WILDCARDS SERIES



Contents



Contents

ISSN 0811 - 577X

LIST PRINT Geelong & Werribee.