(continued from page 185)

changes in length.

The program is written so that it will run on a Vic-20 of any size and with changes to the sound routine in line 325 and the screen colour Poke in line 705 will run on the CBM-64. The number of Data satements can be increased on expanded machines provided they are of the same length. As the program does not change in length when data is added it can be re-saved in the same space on the tape. It is considerably more convient to use the program if it is the first program on the tape as it is necessary to Save the program whenever new data is added.

The following is a short explanation of the program lines, more in order of execution than the way they are listed.

Lines 700 to 730 look for the start of Basic and print the menu. The option selected is input in line 740 and is checked for validity, the appropriate subroutine is selected in line 750.

The search option is handled in line 390 but most of the work is done in lines 330 to 380. Line 330 reads the Data statements and look for an asterisk. The check for a match with the search string is done in line 340. The subroutine call to line 310 checks for a full screen of records. If the screen is full this in turn calls the subroutine in line 300 which suspends operations until Return is pressed. Line 360 prints 'That's all' when the search is exhausted. Line 370 produces the "No Record Found" message and sounds the tone — line 325. Lines 400 to 430 print out all the records, checking for full screens and waiting at the end of the list for another Return. Lines 440 to 570 handle data input. The size of the list is checked in line 440 to see if there is enough room for another entry. Lines 450 to 490 input the information to be stored and fill the spaces with hyphens.

The check for C = 4 — option 4 — is present as lines 450 to 490 are used again in the editing option. The first empty record is found in lines 500 to 520 and lines 530 calculates the address to start Poking the data into place n line 540. Lines 560 and 570 do the same for the first data statement — line 200 — which is the number of records + 1000. The edit option is managed in lines 580 to 620 mostly by using parts of other routines, this allows a lot of program to be squeezed into a small space. Lines 630 and 640 end the program, if data has been added or altered line 640 prompts for the tape to be rewound and then does the Save.

Lines 5 to 260 must be typed in exactly as in the listing — including the Rem — but with no space between the line number and the first character of data. The data in lines 202 to 260 consist of twenty two asterisks. When the program is Run your input data will appear in these lines.

A model universe

W Roy Masefield,
Holland-on-Sea,
Essex.

SPECTRUM

THE MOTION of two bodies under the influence of gravity — the so-called two-body problem of classical mechanics — lends itself very well to demonstration by computer. The mathematics of the problem is quite straightforward, involving nothing more than Newton's Laws of Force and Motion. The possible demonstrations include such things as the capture of one astronomical body by another; orbits of satellites; behaviour of binary stars and so on, given only the starting data.

When the program is Run, the data is called for, and to give you some idea of suitable values to start with, I suggest you try out those given in figures 1 to 5. Then, when you get the feel of it, to try your own values. Angles, by the way, are entered in degrees — the program converts these to radians as required by the ZX-Spectrum — with zero being the east direction. Negative values for the angles are legitimate.

The initial positions of the two bodies are entered as x and y co-ordinates, these being the normal pixel co-ordinates of the Spectrum, but do not go above x=170 because the right-hand quarter of the screen is

(continued on page 189)

Figure 1.

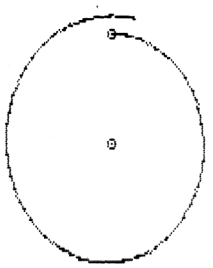
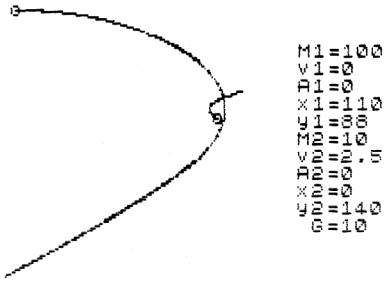


Figure 2.




```
   5 REM "MU" Model Universe
   6 REM © W.R.Masefield 1983
  10 REM Input Data
  20 CLS : PRINT "MODEL UNIVERSE
": PRINT
  30 INPUT "First Body: Mass? ";
m1,"Velocity? ";v1,"Angle? ";a1,
"x coord? ";x1,"y coord? ";y1,"S
econd Body: Mass? ";m2,"Velocity
? ";v2,"Angle? ";a2,"x coord? ";
x2,"y coord? ";y2,"Gravitational
 Constant? ";g: CLS
  40 PRINT TAB 23;"M1=";m1: PRIN
T TAB 23;"V1=";v1: PRINT TAB 23;
"A1=";a1: PRINT TAB 23;"X1=";x1:
 PRINT TAB 23;"Y1=";y1: PRINT TA
B 23;"M2=";m2: PRINT TAB 23;"V2=
";v2: PRINT TAB 23;"A2=";a2: PRI
NT TAB 23;"X2=";x2: PRINT TAB 23
;"Y2=";y2: PRINT TAB 24;"G=";g
  50 LET dt=0.1: LET a1=a1*PI/18
0: LET a2=a2*PI/180: LET x1=x1+1
0: LET x2=x2+10
  60 CIRCLE x1,y1,2: CIRCLE x2,y
2,2
  70 LET v1x=v1*COS a1: LET v2x=
v2*COS a2
  80 LET v1y=v1*SIN a1: LET v2y=
v2*SIN a2
 100 REM Processing
 110 LET fg=g*m1*m2/((x2-x1)*(x2
-x1)+(y2-y1)*(y2-y1))
 120 IF x2=x1 THEN LET phi=PI/2:
 GO TO 140
 130 LET phi=ATN ABS ((y2-y1)/(x
2-x1))
 140 LET f1x=fg*COS phi*SGN (x2-
x1): LET f2x=-f1x
 150 LET f1y=fg*SIN phi*SGN (y2-
y1): LET f2y=-f1y
 160 LET a1x=f1x/m1: LET a2x=f2x
/m2
 170 LET s1x=v1x*dt+a1x*dt*dt/2:
 LET s2x=v2x*dt+a2x*dt*dt/2
 180 LET v1x=v1x+a1x*dt: LET v2x
=v2x+a2x*dt
 190 LET a1y=f1y/m1: LET a2y=f2y
/m2
 200 LET s1y=v1y*dt+a1y*dt*dt/2:
 LET s2y=v2y*dt+a2y*dt*dt/2
 210 LET v1y=v1y+a1y*dt: LET v2y
=v2y+a2y*dt
 300 REM Plot
 310 PLOT x1,y1: DRAW s1x,s1y: P
LOT x2,y2: DRAW s2x,s2y
 320 LET x1=x1+s1x: LET x2=x2+s2
x
 330 LET y1=y1+s1y: LET y2=y2+s2
y
 340 GO TO 100
```

(continued from page 187)

reserved for displaying the starting data. When all the data is entered, the program starts to plot the paths. It is not fast, but fascinating to watch. Plotting continues until one of the curves runs off the edge of the screen area, but plotting can be stopped at any point by Break.

A look, now, at some of the possible demonstrations:

Figure 1: This shows a small mass orbiting a much larger one.

Figure 2: In this, a small body approaches a more massive body at an oblique angle to the direction of the gravitational force. The larger body is initially at rest. The smaller body is pulled out of its straight-line path and traces a near-parabola, and the larger body is given some motion and it begins to trace out a curved path.

Figure 3: Binary stars are illustrated here. One star is larger than the other, and each has been given the required tangential velocity for a circular orbit round the common centre of gravity. Try other velocities and see what happens:

Figure 4: This is an Earth-Moon simulation. The larger body is given a velocity as if in orbit round the sun. As we are looking at only a short piece of this orbit, the lack of curvature does not matter. The smaller mass is given the velocity calculated to keep it in orbit, plus the velocity of the larger.

The resultant paths are fair approximations to what actually happens with the Earth-Moon system; the moon performs a series of loops and the Earth, because of the mutual gravitational action, does a series of cycloid-like loops. Certainly, the Earth's path is not a smooth one.

Figure 5: For fun, we may see what happens when the force of gravity is repulsive instead of attractive, i.e., when we have the anti-gravity of science-fiction.

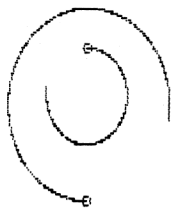
To do this, enter a negative value for the gravitational constant. The figure shows two equal masses approaching each other almost, but not quite, in a direct line. See if you can interpret what is happening if they *are* in an exact head-on course.

Adventurous programmers might like to develop this program further by including action under electric forces. In this case, instead of F_g, we shall have F_E, which depends on the electric charges (q) carried by the bodies:

$$F_E = \frac{kq_1q_2}{d^2}$$

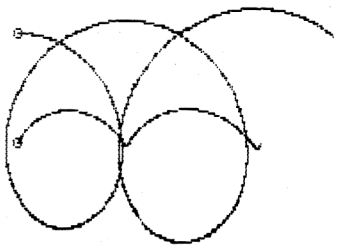
The motions resulting from this force behave exactly as for gravitation, i.e., they depend on the masses of the bodies.

Figure 3.



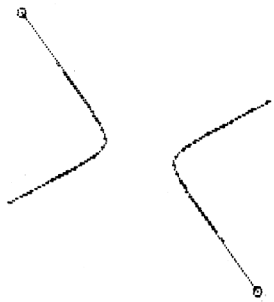
```
M1=100
V1=1.49
A1=0
X1=80
Y1=125
M2=50
V2=2.98
A2=180
X2=80
Y2=55
 G=10
```

Figure 4.



```
M1=100
V1=12.16
A1=0
X1=0
Y1=130
M2=600
V2=0.33
A2=0
X2=0
Y2=80
 G=10
```

Figure 5.



```
M1=100
V1=4
A1=-45
X1=0
Y1=150
M2=100
V2=4
A2=135
X2=140
Y2=30
 G=-7
```

Connect Four

A Cregeen,
Braddan,
Isle of Man.

BBC

IN CONNECT FOUR for the BBC Model B or Electron the idea is to create a row of four counters. Full instructions are given in the program. It plays a pretty mean game and you will have to be on your toes to outwit it. Average response time is eight seconds. A help facility is included if you are completely stuck.

```
   10*TV 255
   20MODE7
   30PROCINSTRUCTIONS
   40DIM A%(7,6),P%(7,6),C%(7),N%(7),CHOOSE(5),CHOOSEY
(5)
   50MODE5
   60PROCDRAWBOARD
   70PROCENTERMOVE
   80IF H$="HELP" THEN GOTO 130
   90PROCFINDY
  100PROCWIN(H%,R%)
  110PROCDRAWCOUNTER(H%,R%,1)
  120IF WIN%=1 THEN PRINTTAB(0,28);"You win!
                    ":PROCNEWGO:RUN
  130PRINTTAB(0,28);"Thinking..."
  140PROCMYMOVE
  150IF H$="HELP" THEN PRINTTAB(0,26);"Try column '";U
%;"':GOTO70
  160PROCDRAWCOUNTER(U%,I%,2)
  170IF P%(U%,I%)>900 THEN PRINTTAB(0,28);"I WIN !
            ":PROCNEWGO:RUN
  180PRINTTAB(0,28);"
      "
  200RY%=0
  210FOR AS%=1 TO 7:IF A%(AS%,1)<>0 THEN RY%=RY%+1
  220NEXT
  230IF RY%=7 THEN PRINTTAB(5,15);"D R A W":PROCNEWGO:
RUN
  235FOR VB%=1 TO 7:FOR FD%=1 TO 6:P%(VB%,FD%)=0:NEXT,
  240GOTO70
  250DEFPROCDRAWBOARD
  260GCOL0,3
  270FOR F%=0 TO 7
  280MOVE (F%*181),1024
  290DRAW (F%*181),256
  300IF F%<>7 THEN PRINTTAB((F%*3),25);F%+1
  310NEXT
  320FOR F%=0 TO 6
  330MOVE 0,1020-(F%*128)
  340DRAW 1270,1020-(F%*128)
  350NEXT
  360ENDPROC
  370DEFPROCENTERMOVE
  380PRINTTAB(0,28);:INPUT"Which column",H$
  390PRINTTAB(0,28);"                    ";TAB(0,26);"
                    "
  400IF H$="HELP" THEN ENDPROC
  410IF ASC(H$)<48 OR ASC(H$)>57 OR LEN(H$)>1 THEN GOT
O 380
  411IF A%((VAL(H$)),1)<>0 THEN GOTO 380
  420H%=VAL(H$)
  430ENDPROC
  440DEFPROCFINDY
  450W%=7:K%=0
  460REPEAT
  470W%=W%-1
  480IF A%(H%,W%)=0 THEN R%=W%:K%=1
  490UNTIL K%=1
  500ENDPROC
  510DEFPROCDRAWCOUNTER(X%,Y%,Z%)
  520GCOL0,Z%
  530IF Z%=2 THEN A%(X%,Y%)=1
  540IF Z%=1 THEN A%(X%,Y%)=2
  550Y%=1020-(Y%*128)
  560X%=X%*181
  570MOVEX%,Y%:MOVE X%-181,Y%:PLOT85,X%,Y%+128
  580PLOT85,X%-181,Y%+128
  590PROCDRAWBOARD
  600ENDPROC
  610DEFPROCMYMOVE
  620FOR X%=1 TO 7:FOR Y%=1 TO 6
  630M%=2:PROCSCAN(-1,-1,1,X%,Y%)
  640M%=2:PROCSCAN(-1,0,2,X%,Y%)
  650M%=2:PROCSCAN(-1,1,3,X%,Y%)
  660M%=2:PROCSCAN(0,1,4,X%,Y%)
  670M%=2:PROCSCAN(1,1,5,X%,Y%)
  680M%=2:PROCSCAN(1,0,6,X%,Y%)
  690M%=2:PROCSCAN(1,-1,7,X%,Y%)
  700PROCPTS(X%,Y%)
  710NEXTY%,X%
  720PROCCHOOSE
  730ENDPROC
```

(continued on next page)

1K CHESS

Last month David Horne showed you the board. This month he illustrates the move logic.

As a reminder, figure 1 depicts the playing board and figure 2 is the Basic program used to enter the machine code. This month we will start with the main driver. Enter Fast mode and change line 10 to read:

10 FOR A = 16681 or 16779

Run 10 and start entering the code shown in figure 3. The next routine to enter will be the keyboard controller, after typing Save "4". Change line 10 to read:

10 For A = 16860 to 16916

Run 10, start entering the code shown in figure 4 and then Save "5". Enter Slow mode and type Run — be prepared for a crash.

The machine is waiting for you to type either a W or B, and will respond by placing the appropriate symbol towards the top right of your screen. Try entering any letter you like first, to ensure that the code rejects all other inputs.

Now enter a number between 1 and 8, followed by a letter A to H at which point just about anything could have happened. The reason for the above exercise is to determine if the keyboard controller is working, that is, selecting which keys will be accepted as inputs.

At this point, it becomes very much more difficult to provide the reader with a comprehensive set of diagnostics. Suffice to say that we are breaking each party down into small routines, which can be re-entered one routine at a time in an effort to isolate the offending error.

Alternatively you may wish to try the program given in figure 7 of last month's article. Assuming after much murmuring, you appear to have this bit correctly entered, our next routine is the address converter. This takes the alphanumeric input and translates it to a board position. Change line 10 to read:

10 For A = 16801 to 16859

Run 10 and start entering the code shown in figure 5 and Save "6". The address converter has a little routine tacked on to the end which effectively checks to see whether the correct piece is being moved. So now enter Slow mode and Run the program. By the way, it will crash again. Enter your opening colour — B or W — then try spare spaces and the wrong colour positions to ensure that these entries are not acceptable. Now try a correct position, at which point the system will crash.

Finally, the move logic. This is going to be a bit of a marathon. You will be entering 170 numbers, around 15 minutes work. Change line 10 to read:

10 FOR A = 16917 to 17087

Run 10 and start entering the code shown in figure 6, and finally Save "7". Enter Slow mode, Delete lines 10 to 15, and Save "ZXCHESS", type Run and hope!

To help with error corrections look for the following: if the pawns do not move correctly, then look at the code from 17021 to 17087; problems with any other pieces, look at code 16981 to 17020. There is a piece selector from 16949 to 16980, if just one appears incorrect. The move tables are at 16926 to 16941, with a part of the pawn table at 16778 to 16780.

Now for a brief outline covering the important functions of the machine-code routines presented this month. We started at address 16681 with a call to a subroutine — 16860 — which tests for which key has been pressed, returning with a value in "a" which is compared with B(39) or W(60) to see who should start. Any other response will start the test for key depressed sequence again.

Dependent upon B or W being pressed, a black or white square is placed in the "whose move now" position located by routine DP1. The next routine at address 16701 overwrites the From and To data at the bottom of the board in preparation for the new data entry and is effectively the start of the move routine.

© D. Horne. 1983.

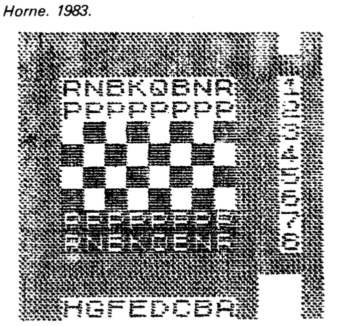


Figure 1. The board.

```
1 REM *** LOTS OF SPACES ***
9 RAND USR 16570
10 FOR A=16514 TO 17087
11 INPUT B
12 POKE A,B
13 PRINT AT 1,0;A
15 NEXT A
```

Figure 2. Machine-code entry program.

```
                 205 220  65 254  39
 40   7 254  60   32 245 175  24
16694
  2  62 128 205  144  65 119  62
  8 205 152  65  119  43 119 205
146  65 119  43  119  35 205 161
 65 254   3  32  234  34  60  64
175  50  62  64   89 205  46  66
16734
205 152  65 205  161  65 254   2
235  48 212  33   62  64  53 126
 60  40 204 133  111 126 185  32
242  42  60  64  126  54   0  18
205 156  64 205  144  65 126 198
16774
126 119  24 179   15  16
```

Figure 3. Main drives.

Continuing, we have a call to subroutine KT, the routine — 16801 — which accepts a number between 1 and 8 followed by a letter between A and H, puts them in the From position and transforms the pair of them to an address on the board by means of the address converter routine — 16806.

The board address is then tested to see whether the contents of the address is either the current mover's piece, an opposition piece, an empty square or part of the backcloth, and, depdenent upon which, "a" is set to a specific number. If it is not one of the current mover's pieces, it returns to the start of the move routine. If it is a current mover's piece, the From position is Saved and the number of legal moves for the position is set to zero.

The piece-move routine — 16942 — is called. This puts all the legal board positions that the From piece can move to into a table. The next data entry is also decoded by subroutine KT. providing a board position, and a status code of that position in "a". The "a" register is interrogated to see if the To

(continued on page 83)

Figure 4. Keyboard controller.

```
                             205 187
  2  62 255 189  40 248 186  40
245  68  77 205 189   7 126 201
213  66  75 197 205 220  65 193
185  40   6  12  16 245 209  24
16894
239 209 209 225 213 119  43 201
229  17  29   8 205 238  65 229
 17  38   8 205 238  65 201
```

Figure 5. Address converter.

```
            229 205   6  56 225
126 214  28  71  14  15 175 129
16814
 16 253 198  31  43  70 144 205
140  65  79   9 126 229 254 128
  6   1  40  23 254   0  40  19
  4 254   8  40  14   4 230 128
197 205 144  65  78 185 193  40
16854
  2   6   0 120 225 201
```

Figure 6. Move logic.

```
                             33
 62  64  52 126 133 111 113 201
  1  15 255 241 242 240  16  14
16934
 17 239  29 227 225  31 243  13
126 230 127 254  53  40  72  14
  1   6   8  33  38  66 254  51
 40  21  33  30  66 254  48  40
 14  72 254  54  40   9   6   4
16974
254  55  40   3  33  34  66 123
134 245 229 197 203 127  32  24
205 181  65 254   2  48  17 205
 21  66 254   0  40  10 193 225
121 254   1  40   5 241  24 224
17014
193 225 241  35  16 217 201 126
230 128  40   5  33  35  66  24
  3  33 140  65  22   3 123 134
229 245 203 127  32  24 205 181
 65 254   1  32  24 122 254   1
17054
 32  12 205  21  66 123 254  30
 56  19 254  90  48  15 241 225
 43  21  32 218 201 122 254   1
196  21  66  24 241 241 225  95
 24 205
```

D. Horne. 1983.

(continued from page 81)

board position is empty or has an opposition piece present. If neither of these two is true the code returns to the start of move routines. If true, the board address is compared — 16745.

With each of the legal move positions tabulated by the From routine. Absence of a match restarts the routine again. A match leads on to the move routine — 16759 — which recalls the From position, gets its contents and replaces them with zero, the contents are then put in the 'To' position and routine B/W called. This rewrites all the black vacant squares and finally the "whose move now" square is changed followed by a jump back to the start-of-the-move routine.

The listing provides for a double move for pawns on their initial set-up ranks. The piece moves are controlled by the tables — 16926 — which are called — 16949 — to permit the appropriate movement associated with that piece. See figure 7.

Figure 7.

```
16681 CALL TKP        205 220 65
      CP N            254 39
      JP Z DIS        40  7
      CP N            254 60
      JP NZ DIS       32  245
      XOR A           175
      JP DIS          24  2
      LD A N          62  128
      CALL DPI        205 144 65
      LD (HL) A       119
16701 LD A N          62  8   (Move
                              Routine)
      CALL DP5        205 152 65
      LD (HL) A       119
      DEC HL          43
      LD (HL) A       119
      CALL DP4        205 148 65
      LD (HL) A       119
      DEC HL          43
      LD (HL) A       119
      INC HL          35
      CALL KT         205 161 65
      CP N            254 3
      JP NZ DIS       32  234
      LD (NN) HL      34  60  64
      XOR A           175
      LD (NN) A       50  62  64
      LD E C          89
16731 CALL MOVE       205 46  66
      CALL DP5        205 152 65
      CALL KT         205 161 65
      CP N            254 2
      EX DE HL        235
      JP NC DIS       48  212
      LD HL NN        33  62  64
      DEC (HL)        53
      LD A (HL)       126
      INC A           60
      JP Z DIS        40  204
      ADD L           133
      LD L A          111
      LD A (HL)       126
      CP C            183
      JP NZ DIS       32  242
      LD HL (NN)      42  60  64
      LD A (HL)       126
      LD (HL) N       54  0
      LD (DE) A       18
      CALL B/W        205 156 64
      CALL DPI        205 144 65
      LD A (HL)       126
      ADD N           198 128
      LD (HL) A       119
      JP DIS          24  179
                      15  16
16801 PUSH HL         229     (KT)
      CALL KYBD       205 6   66
      POP HL          225
16806 LD A (HL)       126     (Address
                              Converter)
      SUB N           214 28
      LD B A          71
      LD C N          14  15
      XOR A           175
      ADD A C         129
      DJNZ DIS        16  253
      ADD N           198 31
      DEC HL          43
      LD B (HL)       70
      SUB B           144
16821 CALL DPZ        205 140 65 (Test)
      LD C A          79
      ADD HL BC       9
      LD A (HL)       126
      PUSH H L        229
      CP N            254 128
      LD B N          6   1
      JP Z DIS        40  23
      CP N            254 0
      JP Z DIS        40  19
      NC B            4
      CP N            254 8
      JP Z DIS        40  14
      INC B           4
      AND N           230 128
      PUSH BC         197
      CALL DPI        205 144 65
      LD C (HL)       78
      CP C            185
      POP BC          193
      JP Z DIS        40  2
      LD B N          6   0
      LD A B          120
      POP HL          225
      RET             201
16860 CALL NN         205 187 2  (TKP)
      LD A N          62  255
      CP L            189
      JP Z DIS        40  248
      CP H            188
      JP Z DIS        40  245
      LD B H          68
      LD C L          77
      CALL NN         205 189 7
      LD A (HL)       126
      RET             201
16878 PUSH DE         213
      LD B D          66
      LD C E          75
      PUSH BC         197
      CALL NN         205 220 65
      POP BC          193
      CP C            185
      JP Z DIS        40  6
      INC C           12
      DJNZ DIS        16  245
      POP DE          209
      JP DIS          24  239
16895 POP DE          209
      POP DE          209
      POP HL          225
      PUSH DE         213
      LD (HL) A       119
      DEC HL          43
      RET             201

16902 PUSH HL         229     (KYBD)
      LD DE NN        17  29  8
      CALL NN         205 238 65
      PUSH HL         229
      LD DE NN        17  38  8
      CALL NN         205 238 65
      RET             201
16917 LD HL NN        33  62  64
                              (ADDLIST)
      INC (HL)        52
      LD A (HL)       126
      ADD L           133
      LD L A          111
      LD (HL) C       113
      RET             201
16926 1 15 -1 -15 -14 -16 16 14
16934 17 -17 29 -29 -31 31 -13 13
16942 LD A (HL)       126     (Piece
                              move)
      AND N           230 127
      CP N            254 53
      JP Z DIS        40  72
      LD C N          14  1
      LD B N          6   8
      LD HL NN        33  38  66
      CP N            254 51
      JP Z DIS        40  21
      LD HL NN        33  30  66
      CP N            254 48
      JP Z DIS        40  14
      LD C B          72
      CP N            254 54
      JP Z DIS        40  9
      LD B N          6   4
      CP N            254 55
      JP Z DIS        40  3
      LD HL NN        33  34  66
16981 LD A E          123     (Piece
                              move
                              routine)
      ADD A (HL)      134
      PUSH AF         245
      PUSH HL         229
      PUSH BC         197
      BIT 7A          203 127
      JP NZ DIS       32  24
      CALL NN         205 181 65
      CP N            254 2
      JP NC DIS       48  17
      CALL ADDLIST    205 21  66
      CP N            254 0
      JP Z DIS        40  10
      POP BC          193
      POP HL          225
      LD A C          121
      CP N            254 1
      JP Z DIS        40  5
      POP AF          241
      JR DIS          24  224
      POP BC          193
      POP HL          225
      POP AF          241
      INC HL          35
      DJNZ DIS        16  217
      RET             201
17021 LD A (HL)       126     (Pawn
                              move
                              routine)
      AND N           230 128
      JP Z DIS        40  5
      LD HL NN        33  35  66
      JR DIS          24  3
      LD HL NN        33  140 65
      LD D N          22  3
      LD A E          123
      ADD A (HL)      134
      PUSH HL         229
      PUSH AF         245
      BIT 7 A         203 127
      JP NZ DIS       32  24
      CALL NN         205 181 65
      CP N            254 1
      JP NZ DIS       32  24
      LD A D          122
      CP N            254 1
      JP NZ DIS       32  12
      CALL ADDLST     205 21  66
      LD A E          123
      CP N            254 30
      JP C DIS        56  19
      CP N            254 90
      JP NC DIS       48  15
      POP AF          241
      POP HL          225
      INC HL          43
      DEC D           21
      JP NZ DIS       32  218
      RET             201
      LD A D          122
      CP N            254 1
      CALL NZ ADDLST  196 21  66
      JR DIS          24  241
      POP AF          241
      POP HL          225
      LD E A          95
      JR DIS          24  205
```

Just when Earth thought its defences were secure the invaders discover machine code. Now Chris Mortimer passes on the secret to 16K ZX-81 owners.

SPACE IN

THIS GAME has 40 deadly manoeuvring Invaders, three laser bases, five defence shields, increasing speed, and an on-screen updated score written completely in machine code to ensure fast and exciting action. The machine code is stored in one long Rem statement which is 810 characters long. Due to the fact that a Rem of this length would be hard to enter direct from the keyboard, try it this way. First, enter the first line:

```
1 REM @ 94 characters @
```

Second, create lines 2 to 8 by editing line 1. Third, enter line 9

```
9 REM @ 10 characters @
```

Four,

```
POKE 16512,3
POKE 16511,44
```

Fifth, enter the rest of the program in listing 1. Listing 2 contains the hex dump of the actual program. This listing is in hexadecimal, explained in chapter 24 of the ZX manual.

It is divided up into sections of 100 units, each preceded by the address of the first unit in both decimal and hexadecimal. There are two ways to start to enter a section of the machine code. To enter the start address in hex, run the program. To enter the start address in decimal you must use the following procedure:

```
LET X EQUAL THE ADDRESS REQUIRED
GOTO 50
```

This will have the same effect as entering the address in hex and will be especially useful when it comes to debugging the program.

Any number of the units may be entered at one time but they must be entered as pairs and not single characters. If the program ends with an error code 5/100 it just means that the screen has been filled, but it is essential that Cont is operated or the last units entered may be lost. The program may be saved at any time as long as the next part to be entered is remembered. Otherwise it may take some time to find where you last got to. It is advisable to save the program at several points in case anything goes wrong.

Once all the program has been entered, Save a couple of copies then you are ready to try it out. Type:

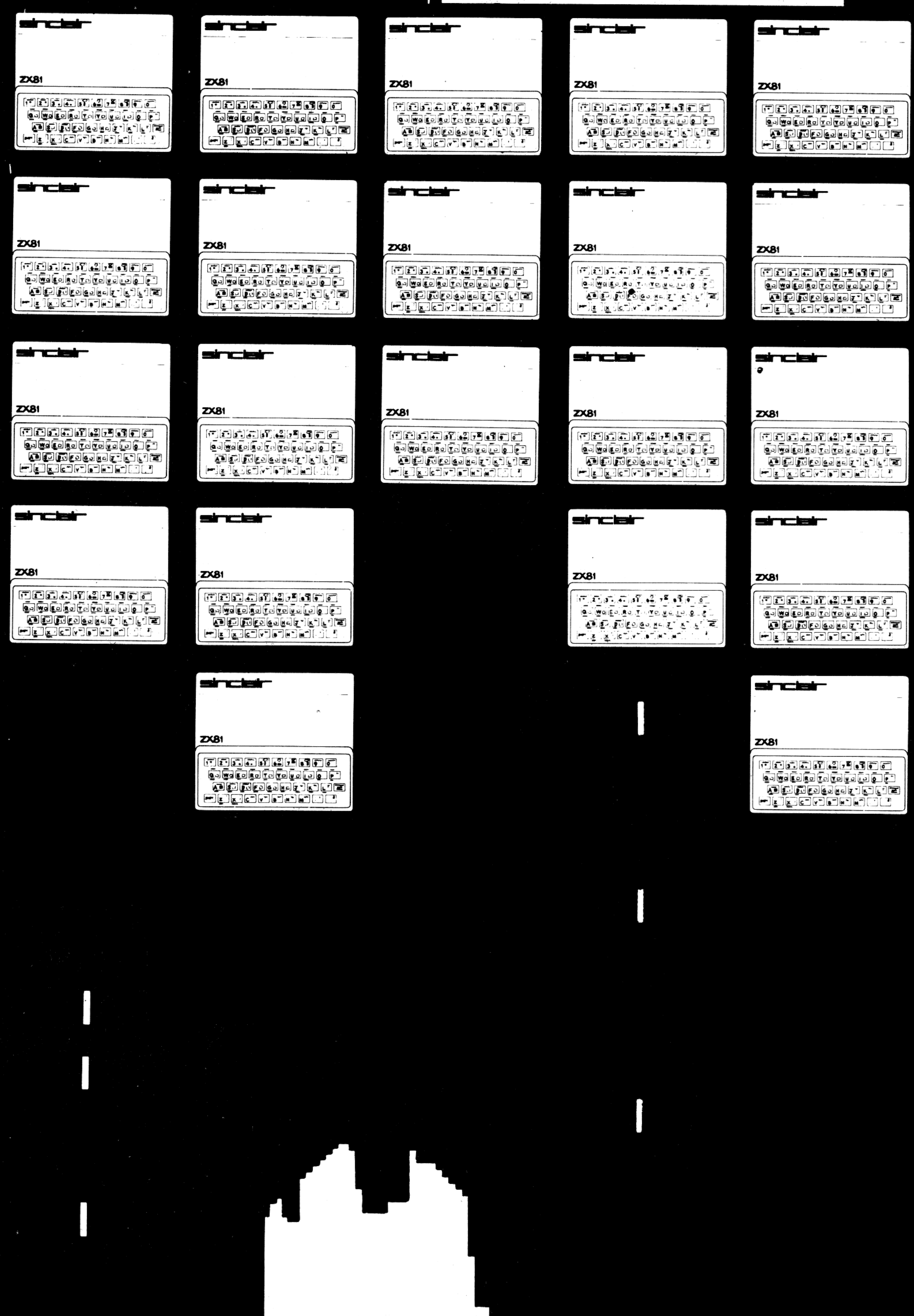
```
RAND USR 17247
```

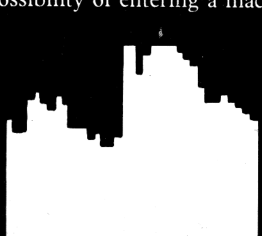
This will jump to the start of the machine code and should produce a picture on the screen just like that of the space invaders. If it does, then congratulations. You have managed the near impossibility of entering a machine-code program correctly first time. If not, then I hope that the following information will help you debug it.

First, though, here is how to change the program into its final form. The final form of the program is made by deleting all the lines except the first — it is important that you do not even edit this line as it will result in some code being lost. This is done by entering the line number of the line you wish to replace and then pressing Newline. New must not be

Listing 1.

```
   1 REM @@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@
   2 REM @@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@
   3 REM @@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@
   4 REM @@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@
   5 REM @@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

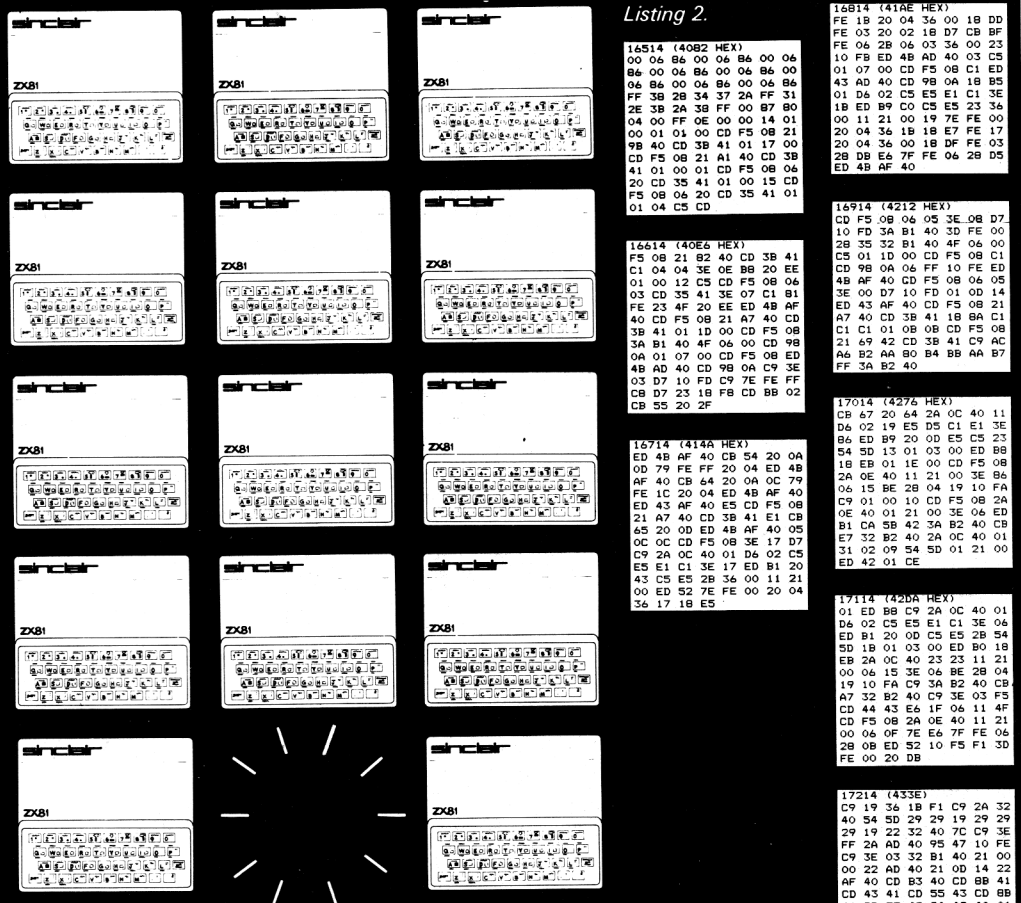




NVADERS

```
0000000000000000000000000000000000
0000000
    5 REM 0000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000
    7 REM 0000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000
    9 REM 0000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000
   9 REM 1234567890
  10 PRINT "WRITE TO  ";
  20 INPUT A$
  30 PRINT A$
  40 GOSUB 200
  50 PRINT
  60 LET A$=""
```

```
  70 IF A$="" THEN INPUT A$
  80 IF A$="S" THEN STOP
 100 PRINT A$( TO 2)
 110 POKE X,16*CODE A$+CODE A$(2
)-476
 120 LET X=X+1
 130 LET A$=A$(3 TO )
 140 GOTO 70
 200 LET X=4096*CODE A$+256*CODE
 A$(2)+16*CODE A$(3)+CODE A$(4)-
122332
 210 RETURN
 300 PRINT "ADDRESS ";
 310 INPUT A$
 320 PRINT A$
 330 GOSUB 200
 340 LET C=PEEK X
 350 PRINT CHR$ INT (C/16+28);CH
R$ (28+C-16*INT (C/16));X
 360 LET X=X+1
 370 GOTO 340
```



Listing 2.

```
16514 (4082 HEX)
00 06 86 00 06 86 00 06
86 00 06 86 00 06 86 00
06 86 00 06 86 00 06 86
FF 38 28 34 37 2A FF 31
2E 3B 2A 38 FF 00 87 80
04 00 FF 0E 00 00 14 01
00 01 01 00 CD F5 08 21
9B 40 CD 3B 41 01 17 00
CD F5 08 21 A1 40 CD 3B
41 01 00 01 CD F5 08 21
20 CD 35 41 01 00 15 CD
F5 08 06 20 CD 35 41 01
01 04 C5 CD
```

```
16614 (40E6 HEX)
F5 08 21 B2 40 CD 3B 41
C1 04 04 3E 0E BB 20 EE
01 00 12 C5 CD F5 08 06
03 CD 35 41 3E 07 C1 B1
FE 23 4F 20 EE ED 4B AF
40 CD F5 08 21 A7 40 CD
3B 41 01 1D 00 CD F5 08
3A B1 40 4F 06 00 CD 98
0A 01 07 00 CD F5 08 ED
4B AD 40 CD 98 0A C9 3E
03 D7 10 FD C9 7E FE FF
C8 D7 23 18 F8 CD BB 02
CB 55 20 2F
```

```
16714 (414A HEX)
ED 4B AF 40 CB 54 20 0A
0D 79 FE FF 20 04 ED 4B
AF 40 CB 64 20 0A 0C 79
FE 1C 20 04 ED 4B AF 40
ED 43 AF 40 E5 CD F5 08
21 A7 40 CD 3B 41 E1 CB
65 20 0D ED 4B AF 40 05
0C 0C CD F5 08 3E 17 D7
C9 2A 0C 40 01 D6 02 C5
E5 E1 C1 3E 17 ED B1 20
43 C5 E5 2B 36 00 11 21
00 ED 52 7E FE 00 20 04
36 17 18 E5
```

```
16814 (41AE HEX)
FE 1B 20 04 36 00 18 DD
FE 03 20 02 18 D7 CB BF
FE 06 2B 06 03 36 00 23
10 FB ED 4B AD 40 03 C5
01 07 00 CD F5 08 C1 ED
43 AD 40 CD 98 0A 18 B5
01 D6 02 C5 E5 E1 C1 3E
1B ED B9 C0 C5 E5 23 36
00 11 21 00 19 7E FE 00
20 04 36 1B 18 E7 FE 17
20 04 36 00 18 DF FE 03
28 DB E6 7F FE 06 28 D5
ED 4B AF 40
```

```
16914 (4212 HEX)
CD F5 08 06 05 3E 08 D7
10 FD 3A B1 40 3D FE 00
28 35 32 B1 40 4F 06 00
C5 01 1D 00 CD F5 08 C1
CD 98 0A 06 FF 10 FE ED
4B AF 40 CD F5 08 0E 05
3E 00 D7 10 FD 01 0D 14
ED 43 AF 40 CD F5 08 21
A7 40 CD 3B 41 18 BA C1
C1 C1 01 0B 0B CD F5 08
21 69 42 CD 3B 41 C9 AC
A6 B2 AA B0 B4 BB AA B7
FF 3A B2 40
```

```
17014 (4276 HEX)
CB 67 20 64 2A 0C 40 11
D6 02 19 E5 D5 C1 E1 3E
86 ED B9 20 0D E5 C5 23
54 5D 13 01 03 00 ED B8
1B EB 01 1E 00 CD F5 08
2A 0E 40 11 21 00 3E 86
06 15 BE 28 04 19 10 FA
C9 01 00 10 CD F5 08 2A
0E 40 01 21 00 3E 06 ED
B1 CA 5B 42 3A B2 40 CB
E7 32 B2 40 2A 0C 40 01
31 02 09 54 5D 01 21 00
ED 42 01 CE
```

```
17114 (42DA HEX)
01 ED B8 C9 2A 0C 40 01
D6 02 C5 E5 E1 C1 3E 06
ED B1 20 0D C5 E5 2B 54
5D 1B 01 03 00 ED B0 18
EB 2A 0C 40 23 23 11 21
00 06 15 3E 06 BE 28 04
19 10 FA C9 3A B2 40 CB
A7 32 B2 40 C9 3E 03 F5
CD 44 43 E6 1F 06 11 4F
CD F5 08 2A 0E 40 11 21
00 06 0F 7E E6 7F FE 06
28 0B ED 52 10 F5 F1 3D
FE 00 20 DB
```

```
17214 (433E)
C9 19 36 1B F1 C9 2A 32
40 54 5D 29 29 19 29 29
29 19 22 32 40 7C C9 3E
FF 2A AD 40 95 47 10 FE
C9 3E 03 32 B1 40 21 00
00 22 AD 40 21 0D 14 22
AF 40 CD B3 40 CD BB 41
CD 43 41 CD 55 43 CD 8B
41 CD 73 42 2A 0C 40 01
D6 02 3E 06 ED B1 28 03
CD B3 40 CD 44 43 E6 03
3C 47 C5 CD 17 43 C1 10
F9 18 D2 EA
```

```
17314 (43A2 HEX)
1D 1E 1F 20 21 22 23 24
25 1C 76
```

used as it will delete line 1. Once this is done line 10 is entered:

10 RAND USR 17247

The program is now complete and can be saved. It is advisable to keep a copy of the program with the hex loader attached as several modifications are suggested later. The program can now be started by the command Run. The command keys are:

W — left
R — right
0-6 — fire

The hex loader listed in listing 1 has an advantage over some hex loaders because it includes a hex lister from 300. Like the hex loader it will accept starting addresses in either decimal or hexadecimal, except that the address to Goto, if decimal is used, is 340.

If you own a ZX Printer you may find it easier to track down a fault if you take a listing and compare it with that in listing 2. Any difference in the Rem statement is a fault, but it should be possible to approximate where in the program the fault is from its position in the Rem.

For the rest, the only way to do it is to slowly search through the program comparing it with the listing. The hex lister will give the address of the unit being looked at in decimal. Knowing this it will be possible to correct any errors by using the hex loader and addressing it in decimal as I explained earlier.

The program is built up of several sub-routines which are called in turn by the main routine, the Driver. These subroutines are: Screen — 40B3 — this routine prints the screen and invaders. Any fault in the screen layout means there is an error here. Keybd — 4143 — this is the routine which finds out which key you are pressing and responds by moving the base or firing a missile.

MVMIS — 418B — this routine moves the missiles up and the bombs down and checks if they have hit anything and adjusts everything accordingly. If the bombs do not move, or the program crashes when the bombs reach the ground then the error is in this routine.

MVINV — 4273 — this routine moves the invaders across and down the screen. If the program crashes when the invaders reach the edge of the screen or disappear on reaching the bottom then the error is here.

DELAY — 4355 — this routine is the one which allows the invaders to increase speed. This is done by repeating a loop a number of times dependent on the score.

Finally, a couple of modifications for those who are never satisfied. The following set of Pokes will reverse the keyboard arrangement:

POKE 16711,109
POKE 16719,92
POKE 16733,76
POKE 16762,93

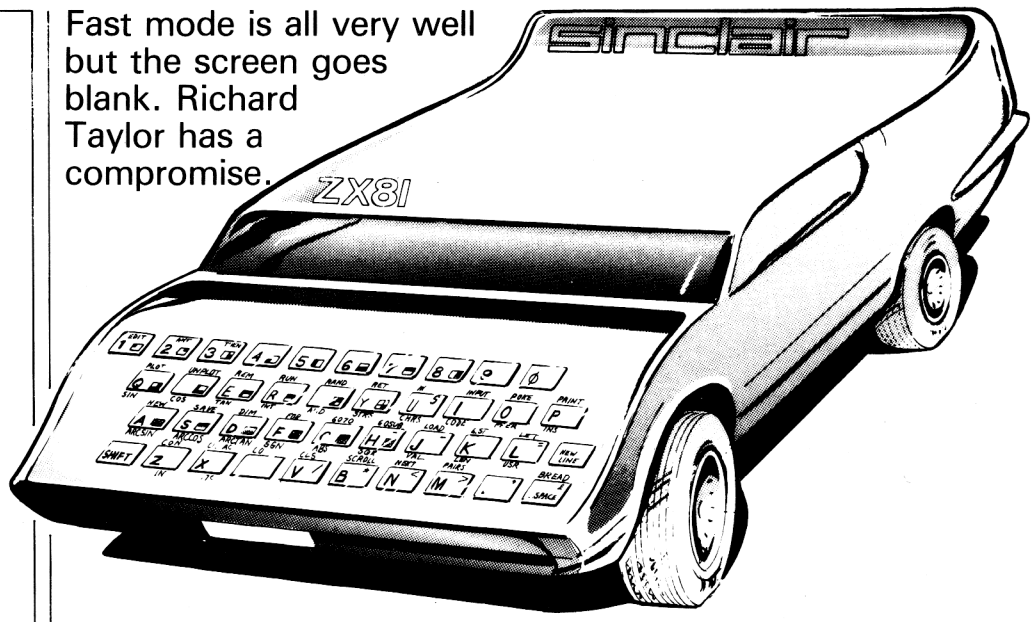
this make the keyboard layout:

P — right
I — left
1-5 — fire

For those who, unlike me, can get very high scores on these games this one can be made much harder by increasing the number of bombs dropped every time. This is done by the following method:

POKE 17301,7 — To double the difficulty
POKE 17301,15 — To multiply the difficulty by five. ■

Fast mode is all very well but the screen goes blank. Richard Taylor has a compromise.



SPEED AND SCREEN

Listing 1.

```
16514-253 203
16516-59 246
16518-205 7
16520-2 221
16522-33 142
16524-64 201
16526-237 95
16528-1 1
16530-25 62
16532-245 205
16534-181 2
16536-205 146
16538-2 205
16540-32 2
16542-221 33
16544-165 64
16546-195 164
16548-2 211
16550-253 58
16552-40 64
16554-198 194
16556-50 40
16558-64 205
16560-146 2
16562-205 32
16564-2 221
16566-33 142
16568-64 195
16570-164 2
```

Listing 2.

```
  1 REM
 10 FOR A=16514
    TO 16571
 20 INPUT B
 30 SCROLL
 40 PRINT A,B
 50 POKE A,B
 60 NEXT A
```

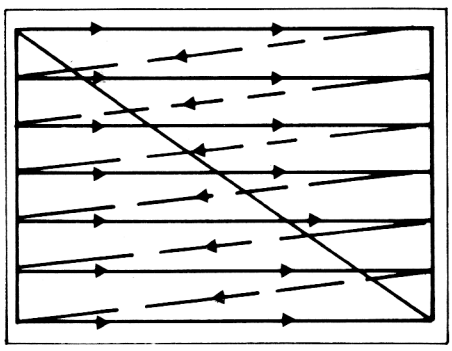


Figure 1. An exaggerated illustration of TV screen scanning. The unbroken line is the active line, the dotted line is the line retrace and the long line from bottom right to top left is the end-of-screen retrace. The lines on the next screen of information will be in the gaps betwen these lines.

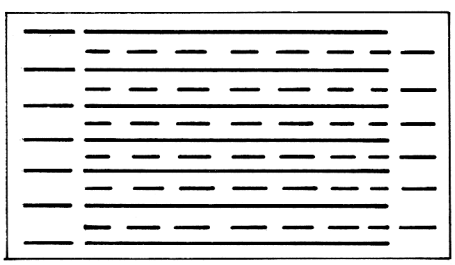


Figure 2. The even lines are transmitted in the first screen of information. The odd — dotted — lines are transmitted in the second screen of information.

THE CULMINATION of this article will be a machine-code program for any memory size ZX-81 which will allow you to run Basic programs in Slow mode at twice the normal speed. It does this by creating an intemediary between Fast and Slow mode. The operating characteristics are similar to those of slow mode except that there is a slight lack of contrast and some screen flicker is introduced.

The program in listing 2 is a hexadecimal loader program for entering the machine code. Line 1 is a Rem statment followed by 70 Xs. The actual machine code is shown in listing 1. Each byte should be entered, followed by Newline. Care should be taken to ensure that each byte is entered into the correct address. When you have finished, delete all lines except the line 1 Rem statement which now holds the machine code. Save a version on tape. Enter:

RAND USR 16514

and Newline. If all is well then characters should appear to be composed of small dots and the screen should flicker slightly. The machine will return to Slow mode if Fast, Slow, Pause, Copy, LList or LPrint are used; it will also return to slow mode if a program line is entered.

A program must contain the Rem statement if it is to use this new mode. The statement

RAND USR 16514

can be used as a program line as well as a direct command. The program has many uses and existing software can easily be converted to use this new facility.

Normally the ZX-81 spends 75 percent of its time on updating the display. My program decreases this to only 50 percent. The consequences of this are that some display flicker is introduced and the display loses some of its contrast. To understand how the program works, it helps to know a bit about how a TV works. A television display is made up of a fast-moving beam of electrons which bombards the phosphor-coated screen and makes it glow. The amount the phosphor

(continued on next page)

(continued from previous page)

glows is proportional to the intensity of the electron beam which is controlled, in this case, by the computer.

The beam moves too fast for the human eye to see its individual movements. The beam travels in an ordered and logical way in order to be sure that it visits every part of the screen. Its movement can be likened to reading a book. It starts at the top left-hand corner and travels horizontally from left to right. When it reaches the extreme right-hand side it flies back to the left hand side, sufficiently below the previous scan line to allow another scan line to be inserted between the first two later.

The beam carries on doing this until it reaches the bottom right-hand corner, then covers the whole screen again, except this time it is filling the gaps between the first set of scan lines. When the beam reaches the bottom right-hand corner again it will return to the top and the entire cycle will be repeated again.

The ZX-81 signal starts at the top by transmitting 55 white lines, followed by 192 lines which contain the actual active screen area. Each character is eight lines high and there are 24 rows of characters: eight times 24 equals 192.

Then there are another 55 white lines. This adds up to 302 lines which is about half of 625 lines — the total number of lines displayed by television receivers in this country. It is not exactly half because some extra blank lines are transmitted while the computer is reading the keyboard — which it does 50 times every second — and incrementing the Frames counter.

Exactly the same picture is transmitted on both the odd and even screens of the signal. The ZX-81 actually executes the program during the 110 blank lines. The rest of the time its services are needed to supervise the television display hardware. If some way of increasing the number of blank lines, without seriously reducing the picture quality, could be found then program would run much faster.

The best method is to only transmit the picture on the even lines and to transmit blank lines during the whole of the odd screen. ■

Disassembled machine-code listing.

```
                SET 6,(IY+59)       This puts the machine into
                CALL 519            SLOW mode
                LD IX,LABEL A       Set interrupt vector to
                                    start of new display routine
                RET                 Return to Basic
LABEL A         LD A,R
                LD BC,6401
                LD A,245            Output the display. This only
                CALL 693            happens on even lines.
                CALL 658
                CALL 544            Read keyboard and
                                    increment frames counter.
                LD IX,LABEL B       Set interrupt vector to start
                                    of display routine for odd lines.
                JP 676               Return to interrupted program.
 LABEL B         OUT 253,A           Makes sure that all future
                                     interrupts will be of the
                                     non-maskable type.
                 LD A,(16424)        Number of blank lines above or
                                     below active screen.
                 ADD A,194           Add an extra 194 lines
                 LD (16424),A        Restores new value
                 CALL 658            Initiates the value into
                                     the system.
                 CALL 544            Read keyboard and increment
                                     frames counter.
                 LD IX,LABEL A       Let next interrupt be to the
                                     even line routine
                 JP 676              Return to interrupted program.
```

COMPRESSION OF TEXT FOR THE ZX-81

ZX-Compander, a text compression routine for the ZX-81 is explained by I F Boulton.

Listing 1.

```
   5 LET TOT=0
   9 REM **FOR LISTING 3 CHANGE
16515,16614 TO 16621,16704
  10 FOR I=16515 TO 16614
  20 SCROLL
  30 PRINT I,
  40 INPUT J
  50 LET TOT=TOT+J
  60 POKE I,J
  65 PRINT J
  70 NEXT I
  80 IF TOT<>10191 THEN PRINT "E
RROR IN DATA INPUT"
  90 REM **FOR LISTING 3 CHANGE
10191 TO 7612 **
```

EVEN WITH A 16K RAMpack the memory space of a ZX-81 is rapidly used up by programs containing large amounts of text. Adventure games and programs with Help messages or long instructions can often not be squeezed in at all. What is needed is a way of compressing text before storage so that it occupies much less memory space and then expanding just those sections required at a particular time, back to their original form.

Information can be compressed insofar as it is predictable; that is if we can find, for instance, sequences of letters that frequently occur in the text, we can code the whole sequence as one character each time it occurs. ZX-Compander works by storing certain pairs of characters in the message as single bytes. This method allows a maximum compression of 50 percent if all the message is composed of those pairs of letters which are being coded.

Each character recognised by the ZX-81 is stored as a single byte in memory. the non-inverted letters, numbers and most of the punctuation and arithmetic symbols have codes in the range 0 to 63. Thus of the 256 possible character codes 192 are not used in normal English, these codes are thus available for defining the pairs of letters we wish to encode.

ZX-Compander operates by searching through the text for any of the 18 characters stored in location 16514 to 16529, when one is found the next character is checked to see if it matches any of the 12 characters stored in locations 16514 to 16525.

If such a match is found the first character of the pair is replaced by a character whose code — in the range 64 to 255 — is calculated from the codes of the letters in the pair. The other member of the pair is then overwritten by the rest of the text being moved one character down in memory. This movement does not actually shorten the whole text string, it leaves the compressed text followed by a number of identical characters.

The character repeated at the end is the last character of the original text — this will in fact always be "$" — the number of repeats being equal to the number of times the message has been coded and moved down in the memory. Thus the text will be shortened by one character for every pair found, while the string remains the same length in memory. The actual compression occurs when the machine-code routine returns to Basic.

The BC register pair — whose contents is returned as the argument of the USR function — contains the length of the compressed text. The original string is thus shortened by a line of the form:

LET A$ = A$(TO USR 16621)

The reason why 16 characters are sought out initially and only a subset of 12 of these is accepted for the second member of the pair is that with only 192 (192=16*12) codes free we cannot uniquely represent 256 (256=16*16) pairs.

Listing 2.

16515	42	57	38	52	46
16520	51	56	45	55	41
16525	49	58	44	40	62
16530	237	91	16	64	27
16535	6	3	19	26	254
16540	13	32	248	16	248
16545	201	26	254	13	192
16550	19	26	27	254	13
16555	201	33	146	64	60
16560	71	43	16	253	126
16565	201	205	146	64	58
16570	123	64	71	5	40
16575	8	19	26	254	13
16580	40	247	24	248	19
16585	26	254	13	200	254
16590	64	56	19	245	230
16595	15	205	172	64	215
16600	241	203	63	203	63
16605	203	63	203	63	205
16610	172	64	215	24	225

The expansion section of the program reverses the above proceedure except that the decode result is printed directly onto the screen. This allows the whole memory to be filled with program and data without having to leave an indefinite amount of space available for holding decompressed text.

The compression which can be achieved using the program depends on the careful matching of the first 12 of the 16 characters to the text being coded — the last four have much less impact but are worth considering. The letters used in the listed program are the 12 most common letters in English — ETAOINSHRDLU — plus the space character and three other common letters — GCY. The space character is placed at the start of the list to speed the search as it is frequently the most common character in a text.

As your text may contain many numbers or symbols the program has been designed so that the only alteration needed to allow different sets of characters to be recognised is for the relevant character codes to be Poked into locations 16514 to 16529.

Another feature is that the character table and decoding routine are held in the first Rem while the encoding routine is held in the second. This allows you to delete the encoding routine once all the text has been compressed, freeing another 89 bytes of precious memory.

Beyond the restriction that your initial text cannot contain characters with codes over 63, there is one further problem; because the variables in the ZX-81 are moved about in memory by the machine's operating system it is necessary to flag the beginning and end of the text string so that the routines can recognise their data. This is done by starting

(continued on page 137)

(continued from page 135)

the string with $$$, separating units within the text that need to be separately retrieved — up to a maximum of 256 items — with $ and terminating the data with $$. For this reason the $ sign may not be used casually in the text, and the $$$ sequence must not occur in other strings in the memory.

Start entering the program by typing a line 1 REM followed by a single space and then 100 zeros and a line 2 REM followed by 84 zeros. The machine-code loader in listing 1 can then be typed in and Run. Carefully enter all the decimal codes in listing 2, addresses are given for every fifth byte to help you check your input.

Then edit line 10 to read

FOR I = 16621 TO 16704

and Run the program again, this time entering the codes in listing 3. This should give you two Rems identical to those shown in listing 4. If all is well delete the loading program and enter the Basic lines from listing 4. It is wise to save the program now before running it, as any mistakes in the machine code will crash the machine.

If there are any mistakes in the routines corrections can be Poked into the Rems; any attempt to Edit them will probably corrupt the code.

The program can now be Run. When prompted enter any string you choose which obeys the rules on allowable characters. If you wish you can separate sections within the text with $ signs. After you hit Newline the text will be encoded. You will then be asked for the item number you require.

This will normally be 1 unless you have put $ separaters in the text. If you enter an item number which does not exist the program will crash.

The item you have requested will now be printed starting at the first free print position on the screen. If the original text is not recovered or the program crashes the routine must be carefully checked using Peek. Once everything is working smoothly you can proceed to use the routines in application programs.

As the routine adds five bytes of $ to the message, plus one byte for each separater, the greatest compression will only be achieved on relatively long sections of text. The routines typically achieve compressions of 30 to 40 percent on a wide range of input, from this and the amount of memory occupied by the decoding routine — 104 bytes — it can be seen that savings in overall memory usage begin to appear when the text is over about 300 characters long. Although the compressed string actually appears longer this is due to certain codes appearing as keywords.

In general your final application program should be developed by first loading the ZX-Compander routines from tape, adding in Basic whatever editing facilities you need to aid the input of your text, entering the text and then compressing it using a line like line 30 in listing 4. Once this has been done the editing section and line 2 can be deleted. When you need to print an item all that is required is that the item number be Poked into 16507, the print position be specified using Print At and the routine called using Rand USR 16566.

The routines are not relocatable due to the numerous subroutine calls using absolute addresses. ■

Listing 3.

16621	205	146	64	19	205
16626	162	64	40	60	33
16631	130	64	26	6	16
16636	190	40	5	35	16
16641	250	24	236	5	197
16646	19	33	130	64	26
16651	6	12	190	40	6
16656	35	16	250	193	24
16661	219	120	198	3	79
16666	6	15	129	16	253
16671	193	126	27	18	213
16676	19	66	75	19	26
16681	2	205	162	64	32
16686	246	209	24	190	205
16691	146	64	6	0	14
16696	4	3	19	205	162
16701	64	32	249	201	

Listing 4.

```
   1 REM  ETAOINSHRDLUGCY GOSUB
?(RND.▀<, RETURN $4 SAVE ( SAVE
 TAN , RETURN $""<,. RETURN $TAN
 S█RNDW?F( CLEAR ?RND?▌C▒<, RETU
RN $C RUN / SAVE <, RETURN $COS
RETURN RND5< PRINT NEW ?LN █RNDN
OT LET ACS ZACS ZACS ZACS ZLN █R
NDNOT / LPRINT
   2 REM LN █RND<LN █RNDCWS▙RND,
▘(█C▌7( IF / GOTO ▌VAL <S▙RND,▘Z
█C▘7( IF AT /<=?LEN ▀?▟?▟( CLEAR
 AT █.>STR$ <PI?<,▘LN █RND4 PLOT
 SGN /█LN █RND▘ :.▀<LN █RND4 RAN
D TAN
   5 REM ZX-COMPANDER BY I SOULT
ON
   6 REM 1 HAYWOOD GRANGE,LITTLE
 HAYWOOD STAFFS.
   7 REM TEL. 0889-882071
   8 PRINT "     ZX-COMPANDER DEMO
NSTRATION"
  10 INPUT A$
  20 LET A$="$$$"+A$+"$$"
  25 LET LENGTH=LEN A$
  30 LET A$=A$( TO USR 16621)
  40 REM DECODING ROUTINE
  50 PRINT "ENTER TEXT ITEM NUMB
ER"
  60 INPUT J
  70 POKE 16507,J
  80 RAND USR 16566
  90 PRINT
 100 PRINT "COMPRESSED BY ";100-
(100*LEN A$/LENGTH);" 0/0"
 110 STOP
```