NILUG NEWS

NATIONAL INDEPENDENT LYNX USER GROUP

Magazine for **LYNX** Users

Volume 1. Issue 3.

NILUG NEWS

VOLUME 1. ISSUE 3.

CONTENTS

ANNUAL SUBSCRIPTIONS and BACK ISSUES

UK	£9.00 per year.
OVERSEAS	£12.00 per year.

(six issues)

Copies of Issue 1 and Issue 2 are still available for £1.50 each.

Send Cheques, payable to NILUG, to:-
53,Kingswood Avenue
Sanderstead
South Croydon
CR2 9DQ.

EDITORIAL

It was very nice to be able to meet some of you at the PCW show and to be able to hear your ideas and views on what NILUG should be doing. It was interesting to see both the 96K Lynx and the 128K. Even more important is the fact that at long last some software is beginning to come through at reasonable prices. QUAZAR have some programs for under a fiver which are good value for money. Also there is Mazeman from Abersoft and Vorlon Invaders from Lynxman. I would like to publish a list of available software in the next issue so if you know of anything perhaps you would let me know (or send in a review).

You should find your membership card enclosed. QUAZAR COMPUTING and LYNXMAN have started the ball rolling by offering a discount to members so don't forget to quote your number when you order products.

```
10 REM It has been suggested to me that you may like to
20 REM submit articles on tape. Type them into a BASIC
30 REM program file with REMs. Then send me a tape of
40 REM your article. I will remove all the line numbers and
50 REM 'REMs' before I publish it.
```

I am still getting technical enquiries from people who obviously haven't read the first page of Issue 2. If you want advice PLEASE, PLEASE read the Technical Enquiries article. One further point that I should have put at the top of the list of requirements was WRITE SO THAT I CAN READ YOUR LETTER! It is amazing the number of people who write so illegibly that I can't read their questions.

Some overseas members have written in but say 'sorry I haven't enclosed an SAE but I can't buy british stamps here'. You can purchase an International Reply Coupon. You send it to me and I can use it to purchase a stamp, so no excuses please.

Robert Poate.

VORLON INVADERS

The city of VORLON is experiencing an asteroid storm. The asteroids are burnt up in the atmosphere. Your enemy has decided to attack using missiles to destroy the atmosphere and guided mines to destroy your defence satellite. When the atmosphere is destroyed the asteroids will destroy Vorlon. A fast action game exploiting the versatile graphics of the LYNX. With numerous fast-moving aliens, missiles and asteroids on the screen it requires tactics as well as good coordination. How well can YOU defend Vorlon?

VORLON INVADERS by LYNXMAN Price £5.00
(10% Discount for NILUG members)
270,Tithe Pit Shaw Lane,
Warlingham, Surrey, CR2 9AQ.

AT LAST
MAZEMAN
ABERSOFT
£4.95
FOR THE LYNX
AT A SINCLAIR PRICE
A fast action machine code game
'....is very accurate and fast' *Which Micro*
ABERSOFT 7 MAESFALLEN, BOW ST, DYFED, SY24 5BA
24hr Ansaphone 0970 828851

Dear Robert

Thank you for your letter and the second edition of NILUG NEWS. This proved a much better magazine for learners like my family and myself. Thank you particularly for the 'Machine Code for Beginners' article - the best I've seen. I still do not understand all that you write, but at least I do have the beginning of illumination!

We enjoyed "Othello", but the display needed improvement. This is how we did it.:-

```
ADD     10 GOTO 100                    55 NEXT J
        15 DEFPROC FRAME               60 MOVE 70,34+J
        20 INK  5                      65 DRAW 166,34+J
        30 FOR J=0 TO 96 STEP 12       70 NEXT J
        40 MOVE 70+J,34                75 PROTECT 0
        45 DRAW 70+J,114               80 ENDPROC
        50 NEXT J                     205 PROC FRAME
```

DELETE lines 240;250 and 260.

At line 130 change C=76 to read C=241.

I think that you will agree it does make it easier to follow the game. The LYNX is a good machine, but I wish Camputers would push it a bit more. And whatever happened to Camsoft? They seem to have vanished with hardly a trace!

With good wishes for the success of NILUG,

Yours sincerely,

Robert White.

Dear Sir

I enclose two articles for publication in NILUG NEWS which I hope you will find suitable. Yours sincerely.

Martin D. Judd.

TAPE HANDLING. Compared with much cheaper micros the Lynx tape handling is very poor. Notably missing is the ability to SAVE machine code and variables from BASIC and the inability to LOAD using LOAD"" to load the first program found. Hopefully these faults will be corrected, and I suggest that automatic TAPE speed selection on loading is not too much to expect.

I will now give details of three ROM routines concerned with tape handling. Suppose, for example, you wish to SAVE the bytes from &9000 to &9FFF. The SAVE routine starts at address &3EFF in ROM. Before jumping to this address the start address of the block to be SAVEd (&9000) and its end address (&9FFF) must be pushed onto the stack. The DE register pair must point to the first quote in the name which is terminated by a final quote. The HL register pair must contain the transfer address where execution will begin on MLOADing. If HL is set to zero then the program will continue where it left off. To SAVE the block form &9000 to &9FFF in BASIC you would use:

```
100 CODE EB 21 00 90 E5 21 FF 9F E5 21 00 00 C3 FF 3E
110 REM "BYTES"
120 CALL LCTN(100),LCTN(110)
```

The MLOAD routine starts at &3F62 in ROM. The DE register pair must point to the name in quotes. To load the bytes saved by the above routine use:-

```
100 CODE EB C3 62 3F
110 REM "BYTES"
120 call LCTN(100),LCTN(110)
```

To change the tape speed from machine code load the A register with the speed required and call the routine at &0D93. For example to set TAPE 5:-

```
100 CODE 3E 05 CD 93 0D C9
110 CALL LCTN(100)
```

CIRCLES. CIRCLE is a function which is sadly missing from the Lynx BASIC. This has probably meant that you have had to resort to the SIN/COS method of drawing circles. Apart from being very slow this method plots some points more than once while leaving gaps in other parts of the curve.

A much better way to draw circles on a raser scan device is to use an incremental circle generator. The one I use is an adaptation of Bresenham's algorithm BRES77. This routine only works out the points on the first 45 degrees of the circle and takes advantage of symmetry to plot the other seven points.

Listing 1 shows the main routine. Add the lines in listing 2 for a demonstration. The first two parameters to the CIRCLE procedure are the X and Y coordinates of the center of the circle and the third is the radius. The routine is fast (for BASIC) and draws a very pleasing circle.

```
LISTING 1
100 DEFPROC PLOT            230   PROC PLOT
110 DOT a+x,b+y             240   IF d<0 THEN LET d=d+4*x+6
120 DOT a+y,b+x             250    ELSE d=d+4*(x-y)+10,y=y-1
130 DOT a+y,b-x             260    LET x=x+1
140 DOT a+x,b-y             270 WEND
150 DOT a-x,b-y             280 PROC PLOT
160 DOT a-y,b-x             290 ENDPROC
170 DOT a-y,b+x                   LISTING 2
180 DOT a-x,b+y                   10 TEXT
190 ENDPROC                       20 LET a=RAND(200)+27
200 DEFPROC CIRCLE(a,b,c)         30 LET b=RAND(200)+27
210 LET x=0,y=c,d=3-2*c           40 LET c=RAND(26)
220 WHILE x<y                     50 PROC CIRCLE(a,b,c)
                                  60 GOTO 20
```

OTHER TIPS. The sound output on the Lynx is not very loud. The speaker signal comes out of the Lynx on pin 4 of the cassette socket. You can put an external speaker across pins 4 and 2 to get a louder sound. I have tried 8 and 4 ohm speakers. Both work although the 4 ohm speaker is far louder. Unfortunately they are more expensive.

Another method for increasing the sound was suggested to me at the PCW show. You remove the LYNX sticker on the left of the machine and stick it on the right. Next you cut out the rectangle where the sticker used to be and put a piece of cloth over the hole. The speaker is right underneath this hole and (apparently) you get about twice as much sound out with this mod.

Many owners complain of loading/saving problems with their cassette. It has been pointed out to me that part of the trouble is due to the phasing of the tape. To overcome this problem one member has suggested resoldering the two leads on the head of your cassette recorder. This is a bit drastic and I prefer to use a small lead with a 3.5mm jack plug on one end and a socket on the other. The leads are connected so that they reverse the signal.

REVIEWS

MAZEMAN from ABERSOFT Price £4.95 By Danny Allen

MAZEMAN is the only release I've seen for the LYNX from Abersoft. It loads on TAPE 0 in about 3 minutes. There is a joystick option and then the game settles into the familiar PAC-MAN theme. There are, however, a few differences. The 'muncher' looks like a man on a bike, and the 'pills' have turned into swords (Aberswords in fact). You have three lives with an extra life after 10,000 points. The graphics are excellent (the 'ghosts' even have little eyes), and after 'eating' a sword, they all simultaneously turn blue for ten seconds, during which time they may be 'eaten'. The programming is very good. The more dots are eaten, the more the ghosts chase you and if you 'eat' a sword they all run away! The control of the 'muncher' is tricky at first, but after a few games the keying becomes easier. During hours of playing, only two slight bugs came to my attention. Firstly at the start of the games, the dot that the ghosts always starts moving from is rubbed out. Secondly, sometimes after 'eating' a sword a ghost plays kamikaze and runs towards you. Overall the program is excellent and very addictive. High scores are not easy to get (my highest is 15,000). One touch that I really liked was that the ghosts run about by themselves after the game has finished. Overall, 8 out of 10.

THE WORM from QUAZAR COMPUTING Price £5.95 By D.J.McQueen

This is a type of centipede game, written almost entirely in BASIC, but has a fast display nevertheless. Full instructions are available on screen if required. The object of the game is to guide Wilberforce the Worm through four scenarios, collecting points by eating flowers but avoiding the walls and his own trail. All four scenes require some nimble finger action on the cursor keys to prevent Wilberforce coming to grief. The game has an infinite number of levels and a high score table. The sound effects are very good, using both Lynx's BEEP and SOUND commands. The graphics are also good although blue and cyan are slightly over-used to create the scenes. Overall an entertaining game with fast moves. Value for money 7 out of 10.

96K LYNX Upgrade By Chris Cytera

It is now possible to have the standard 48K Lynx expanded to 96K RAM, 20K ROM. This can be done for around £90. This review sets out to examine the value of this expansion.

One part of the upgrade is the enlargement of the 16K of workspace RAM to 64K. 37.5K of this is available for BASIC programs. A further 23K can be used for machine code programs and data storage. The second part of the deal is a 4K expansion EPROM. The main feature of this is a comprehensive array of commands to manipulate data in the 23K store, including saving data on cassette. Also included are some graphics commands; CIRCLE (filled and empty circles are possible); CLW (clears window). Miscellaneous features are; error trapping; pre-programmed sounds;parallel and serial printer drivers; and the implementation of LIGHTPEN and JOYSTICK.

A bonus is the replacement of the BASIC 1.0 EPROMS with version 2.0. This features several improvements: the infamous ARCSIN and ARCCOS bugs have been removed;LEFT$ and MID$ now function properly with a null string; the monitor now displays the N and C flags correctly; the DISK command has been replaced by XROM which jumps to E000H to call an external ROM.

It would seem at first sight that the upgrade is well worth while. The massive memory allows scope for large BASIC and machine code programs, as well as extensive data storage. The extra graphics commands, although too few for my liking, certainly make the production of graphics quicker and easier. The curing of certain shortcomings present in BASIC 1.0 is also welcome.

And now for the criticism. First, a trivial point: all new commands have to be preceded by EXT, which is a slight nuisance. Second I regard re-programmed sounds as having nothing more than novelty value; their effect soon wears off. Third, BASIC 2.0 should be offered to EVERYONE, FREE OF CHARGE. The errors in version 1.0 are the fault of Camputers, and they should correct them at their expense. Therefore the inclusion of version 2.0 should not be a part of this upgrade at all. Fourthly, I think that the expansion is a little expensive. The cost of the RAM chips is about £37, while the cost of labour is minimal; the task involves pulling out ten chips, plugging in eleven, cutting three PCB tracks and making two soldered links. This leaves about £50 for a 4K extension to BASIC. Considering that 4K toolkit ROMs for other computers sell for £15-£25, this is rather on the high side.

In conclusion, I would say that the 96K upgrade adds signifigant power to the standard 48K Lynx, but is somewhat overpriced. I also believe that the RAM and EPROM should be offered separately; maybe some people do not need the extra commands, and perhaps others have expanded the RAM themselves and only require the EPROM. All in all, a good upgrade but its marketing needs reviewing.

VORLON INVADERS by LYNXMAN Price £5.00 By Danny Allen

Vorlon Invaders auto runs after about 2-3 minutes loading time. It opens with a screen long description of the game with some graphic effects. The game is more complicated than the usual 'space invaders', but not much. You are using your spaceship to defend the city from alien attack. Some aliens destroy the atmosphere when they touch it. Some come down, drop a mine and then go back up. But most are just asteroids, which are the only ones that you don't get any points for blasting. If your base hits anything then one life is lost. The game ends when all 10 lives have been lost or the atmosphere has been destroyed. After ending the game goes back to the original text and gives a high score. The game contains no sound whatsoever. The graphic are very fast indeed, even when constantly firing and moving at the same time. There is an unlimited amount of bullets that can be fired at the aliens, and no limit to the amount that can be on the screen at the same time. Each alien has its own small explosion, and it has to be small because all the graphics are tiny, varying from lower case x (asteroids) to graphics between a quarter and a half the size of normal characters. There is also something which I found quite pleasant. Instead of pressing the spacebar once for every shot, if you keep the spacebar pressed down, a constant stream of bullets appears, which makes the game much easier. The only bug I noticed was that one alien sometimes got stuck at the top of the screen (nothing to pull your hair out about).

As a conclusion then, this game is the best 'invaders' type I've seen for the Lynx, but could have been made better by the use of bigger graphics, and perhaps a 'Hall of Fame' and some sound effects. Overall value 7 out of 10.

ADVENTURE QUEST from LEVEL 9 Price £9.90 By Axel Cleeremans

I took some months to complete Adventure Quest. Since

I think that I am not particularly stupid, this means it is a complex and challenging game. On the matter of value for money, I was pleased to receive a booklet and an envelope along with the cassette. The booklet sets up the background of the game and tells you how to get it loaded, and the envelope is to be used when you are completely stuck and need a hint. You get the hint by return of post.

Now to the game itself. It takes place in a rather mixed-up fantasy world with monsters and objects coming from many different mythologies. So don't be surprised to find sphinxes living close to dragons, snowmen or vampires! The topography is quite weird too, as you'll have to travel through desert, mountains, lakes, volcanos, etc., in a quest to find the demon lord AGALIAREPT and destroy the source of his power, which is threatening Middle-Earth. This may sound simple but the problem is that you need four stones to enter the demon's tower and to get the stones you have to find out how to use several other objects. This may sometimes prove difficult, but that's all the fun! I liked the idea of a final goal to reach. It is a departure from the usual 'collect treasure to win' game and adds consistency to the scenario. There is only one way to win, but in the meantime a simple object can serve several different useful purposes, which allows you to progress even without doing everything right.

There are two hundred or more locations. The reactions of the monsters is very well described, humorous and quite long. The display is green on black when the Lynx has finished printing the results of the move, but it is almost unreadable while it is being printed because the black paper is replaced by a mixture of coloured dots. Although printing only takes some seconds this can be annoying if you are anxious and can't wait until it is completed. The game commands include all the usual verbs such as INVENTORY, TAKE, NORTH, etc, and can be abbreviated. You can also choose to type full sentences of just two words commands. SAVE and RESTORE are fast and easy to use, which is essential as you only have three lives in a single game.

To sum up, Adventure Quest is a wonderful program, fast, exciting and challenging. If you like adventures then this one is for you!

GOMOKU by A.C. Karsten.

```
1 WINDOW 3,123,5,245
100 GOSUB  LABEL UITLEG
110 GOSUB  LABEL VELD
120 INPUT "Do You want to
start Y/N";R$
130 IF R$="Y" THEN  GOTO
LABEL PERSON
135 LABEL COMP
140 FOR M=0 TO 99
150   LET H(M)=0
160 NEXT M
165 PRINT "Let me think ";
170 PROC DENK(1,0,5,0,90)
180 PRINT "*";
190 PROC DENK(9,4,9,0,50)
200 PRINT "*";
210 PROC DENK(10,0,9,0,50)
220 PRINT "*";
230 PROC DENK(11,0,5,0,50)
240 PRINT "* ";
250 LET S=-1,Q=0,c=2
260 FOR M=0 TO 99
270   IF (I(M)<>0) OR (H(M)<S)
THEN  GOTO 2050
290   IF H(M)>S THEN GOTO 2000
310   LET Q=Q+1
320   IF RAND(Q)<1 THEN  C=M
330 NEXT M
335 LABEL ZET
340 LET X=INT(C/10),Y=C-(X*10)
350 PRINT "I'll do ";X;"-";Y
360 PROC SCHERM(X,Y)
370 LET I(C)=5,N=N+1
380 IF N=100 OR (S=0 AND N>1)
THEN  LET U=2
390 IF S>Y(20) THEN  LET U=1
400 IF U<>0 THEN
GOTO LABEL WIN
405 LABEL PERSON
410 ?"What is your move ";
420 LET X=GETN
430 IF X<48 OR X>57 THEN
GOTO 420
440 LET X=X-48
450 PRINT X;"-";
460 LET Y=GETN
470 IF Y<48 OR Y>57 THEN
GOTO 460
480 LET Y=Y-48,c=6
490 PRINT Y
500 PROC CONTRL
510 PROC SCHERM(X,Y)
520 LET I(A)=1,N=N+1
530 IF N=100 THEN  LET U=2
540 IF H(A)>=Y(4) THEN  U=-1
550 IF U=0THEN GOTO LABEL COMP
560 ELSE  GOTO  LABEL WIN
700 DEFPROC DENK(C,J,j,K,k)
710 LET c=4*C
720 FOR M=J TO j
730   FOR m=K TO k STEP 10
740     LET A=M+m,B=A+c,Z=0
750     FOR P=A TO B STEP C
760       LET Z=Z+I(P)
770     NEXT P
780     LET Q=Y(Z)
790     IF Q<>0 THEN  GOSUB
LABEL TUSSEN
800   NEXT m
810 NEXT M
820 ENDPROC
830 LABEL TUSSEN
840 FOR P=A TO B STEP C
850   LET H(P)=H(P)+Q
860 NEXT P
870 RETURN
999 LABEL UITLEG
1000 DIM I(99),H(99),Y(21)
1001 FOR t=0 TO 99
1002   LET I(t)=0,H(t)=0
1003 NEXT t
1004 FOR t=0 TO 21
1005   LET Y(t)=0
1006 NEXT t
1007 LET N=0
1008 LET U=0
1009 Y(0)=1,Y(1)=3,Y(2)=50,
Y(3)=1000,Y(4)=1000000,Y(5)=10
,Y(10)=200,Y(15)=10000,
Y(20)=1E+8
1010 CLS
1020 PRINT "GOMOKU"
1030 PRINT "The object of the
game is to get"
1032 PRINT "five pieces of
the same colour in"
1034 PRINT "a straight line,
horizontal,"
1036 PRINT "vertical or
diagonal. You play"
1038 PRINT "against your Lynx
(red) and your"
1040 PRINT "colour is yellow.
Type your coordinates"
1042 PRINT "in algebraic
notation <X,Y>!!"
1043 PRINT  @ 3,160;"to
continue press space !"
1044 PRINT @ 3,200;CHR$(94);
CHR$(127);"By A.C & J.P.
NETSRAK SOFTWARE Ltd."
1135 LET t=GETN
1400 RETURN
1500 LABEL VELD
1504 PAPER WHITE
1510 CLS
1520 WINDOW 3,123,225,245
1530 FOR i=0 TO 9
1550   INK RED
1560   MOVE 30+(i*20),0
1570   DRAW 30+(i*20),200
1580   PRINT  @ 18+(i*10),232;
CHR$(24);i;CHR$(25);
1590 NEXT i
1591 MOVE 30+(i*20),0
1592 DRAW 30+(i*20),200
1600 FOR i=0 TO 9
1610   LET r=0+i*20
1620   MOVE 30,r
1630   DRAW 230,r
1640   INK BLUE
1650   PRINT  @ 10,i*10;
CHR$(24);9-i;CHR$(25);
1660   INK RED
1670 NEXT i
1671 MOVE 30,200
1672 DRAW 230,200
1680 VDU 23
1700 RETURN
2000 LET C=M,S=H(C),Q=1
2009 NEXT M
2010 GOTO  LABEL ZET
2050 NEXT M
2060 GOTO  LABEL ZET
3000 DEFPROC CONTRL
3010 LET A=10*X+Y
3020 IF I(A)<>0 THEN GOTO 3040
3030 ELSE  ENDPROC
3040 PRINT "ALREADY OCCUPIED"
3050 GOTO  LABEL PERSON
4000 DEFPROC SCHERM(X,Y,c)
4001 LET x=30+X*20
4004 LET y=180-Y*20
4005 INK c
4006 FOR i=1 TO 18
4011   MOVE x+i,y+1
4013   DRAW x+i,y+19
4013.5   BEEP 200+10*i,50,63
4014 NEXT i
4015 INK 2
4200 ENDPROC
5000 LABEL WIN
5006 VDU 23
5010 IF U=1 THEN ?"I won in ";
5020 IF U=(-1) THEN  PRINT
"You won in ";
5030 IF U=2 THEN  PRINT "It's
a draw after ";
5040 PRINT N;"moves"
5040.5 VDU 30
5041 PAUSE 10000
5050 ?"Want to play again? ";
5060 ?"then press spacebar"
5070 LET a=GETN
5080 IF a<>32 THEN  END
5090 ELSE  RUN
```

Writing Graphics Software for the LYNX.

In the article Output to the Screen (Issue 1) the LYNX's screen was described as being composed of 32 8-bit columns by 248 bytes deep giving a resolution of 256 by 248. It was also pointed out that to display a character in one colour 10 or 20 bytes would need to be displayed on the screen. If three colours were involved then 30 or 60 bytes would be needed. It is possible to read bytes from the screen but due to the number of bytes needed to define characters it would be extremely slow to read the screen ram to find out what was being displayed.

This article describes an alternative technique for organising displays so that you know what is displayed. Basically a map of the screen is kept in user ram. As the display is updated so is the map. Just to establish the technique consider displaying a chess-board. If you numbered each square, starting top left and working down in the same way as you read, then the top squares would be numbered 1 to 8 and the bottom row 57 to 64. Sixty four bytes of user ram could be set aside to hold codes to indicate what chess piece was in which square. As you moved the chess pieces so you would update the 64 bytes. If you wanted to find out what was displayed where you would not read the screen but instead you would examine the codes in the 64 bytes. If you wanted to do a chess program I doubt it would be worth using this technique. It would be better to set up a two dimensional array in BASIC. However when the grid becomes large you would soon run out of ram because BASIC variables need five bytes each.

Since the LYNX's screen is 32 8-bit columns by 248 bytes deep several grids are possible. The following table lists the grids and how much user ram would be needed.

ACROSS		DOWN	RAM		
256	x	248	63,488	or	62K
128	x	124	15872	or about	15.5K
64	x	62	3968	or about	4K
32	x	31	992	or about	1K

Since the 48K LYNX only has about 13.5K of ram available to the user the best resolution that is possible to map using one byte per square is 64x62. Having established the grid the next thing to do is to design a game (or whatever) into the grid. You could try 'Bat 'n Ball' but I chose the 'snake' game where you try to encircle your opponent and force him to bump into something without bumping into anything yourself.

Having decided upon the game the next problen was to work out how to write to each of the squares. This is done in the PROCEDURE called BLOCK. You pass in the X and Y coordinates and the colours you wish to have protected. It then calls one of two machine code routines (lines 100 and 110) which will write to the screen. Finally BLOCK will update the ram by changing the original zero to an &FF. Line 120 contains a machine code routine to zero the video map. You pass in HL set up to the start of the map. In this case it is set to &8000. You could use a BASIC loop and POKE zeros into the map but it is much faster to use machine code. I haven't used RESERVE to set aside the memory. I know that the BASIC program is small and that the map is highly unlikely to be damaged by the stack. The procedure RANDIR will return a random pair of numbers (a & b) to be added to the current X,Y coordinates to generate the location into which the next move may be made, provided that the c parameter is 0. If c is not zero then the numbers returned will be the next in a series of possible moves. The NEXTXY procedure will return the next coordinate (R & r) given the current coordinates (a & b) and the numbers to be added to them (c & b). The rest of the program should be quite straightforward.

```
100 CODE  11 20 00 06 04 C5
3E 0F 0E FF CD 6C 62 C1 19
10 F4 C9
110 CODE  11 20 00 06 04 C5
3E F0 0E FF CD 6C 62 C1 19
10 F4 C9
120 CODE  01 00 10 AF 77 23
0B 78 B1 20 F8 C9
130 RANDOM
140 DIM A(17)
150 LET a=0,b=0,c=0,d=0,R=0,
r=0,P=0,Q=0,p=0,q=0,L=0,U=0
160 FOR I=0 TO 17
170   READ A(I)
180 NEXT I
190 LABEL NEXT.GAME
200 PROTECT 0
210 VDU 1,WHITE,2,BLACK
220 CLS
230 REM INSTRUCTIONS IF REQ
240 CALL LCTN(120),&8000
250 LET X=20,Y=20,x=40,y=40
260 PROC RANDIR(a,b,1)
270 LET P=a,Q=b
280 PROC RANDIR(a,b,0)
290 LET p=a,q=b
300 REM
310 LABEL MOVE.LYNX
320 IF RAND(100)>3 THEN GOTO
 LABEL LM
330 PROC RANDIR(a,b,0)
340 LET p=a,q=b
350 LABEL LM
360 LET J=0,e=1+RAND(4)
370 REM CAN I MOVE ?
380 LABEL NEXT.TRY
390 PROC NEXTXY(x,y,p,q,R,r)
400 IF 0=PEEK(&8000+R+r*64)
THEN  GOTO  LABEL MOVE
410 PROC RANDIR(a,b,e)
420 LET J=J+1,e=e+1,p=a,q=b
430 IF J<>5 THEN  GOTO
 LABEL NEXT.TRY
440 GOTO  LABEL END.GAME
450 LABEL MOVE
460 LET x=R,y=r
470 PROC BLOCK(x,y,5)
480 IF PEEK(&8000+X+Y*64)=
&FF THEN GOTO LABEL END.GAME
490 PROC BLOCK(X,Y,3)
500 LET A=KEYN
510 IF A=0 THEN  GOTO 560
520 IF A=10 THEN  Q=1,P=0
530 IF A=11 THEN Q=-1,P=0
540 IF A=12 THEN P=1,Q=0
550 IF A=22 THEN P=-1,Q=0
560 PROC NEXTXY(X,Y,P,Q,R,r)
570 LET X=R,Y=r
580 GOTO  LABEL MOVE.LYNX
590 REM
600 DEFPROC BLOCK(a,b,c)
610 PROTECT c
620 IF 1=(a MOD 2) THEN  CALL
 LCTN(100),INT(a/2)+b*128
630 ELSE  CALL LCTN(110),
INT(a/2)+b*128
640 POKE &8000+a+b*64,&00FF
650 ENDPROC
660 DEFPROC RANDIR(a,b,c)
670 REM random p & q
 returned in a & b
680 IF c=0 THEN  c=RAND(4)
690 LET c=2*c
700 LET a=A(c),b=A(c+1)
710 ENDPROC
720 DEFPROC NEXTXY(a,b,c,
d,R,r)
730 REM returns next x&y
given p&q
740 LET R=a+c
750 IF R>63 THEN  LET R=0
760 IF R<0 THEN  LET R=63
770 LET r=b+d
780 IF r>61 THEN  LET r=0
790 IF r<0 THEN  LET r=61
800 ENDPROC
810 DATA -1,0,1,0,0,1,0,-1,
-1,0,1,0,0,1,0,-1,-1,0
820 LABEL END.GAME
830 VDU 1,BLACK,2,WHITE
840 PROTECT 0
850 IF J=5 THEN  GOTO 900
860 LET L=L+1
870 PRINT  @ 20,20;"YOU
 CRASHED   ";
880 GOTO 920
890 REM
900 PRINT  @ 20,20;"LYNX
 CRASHED  ";
910 LET U=U+1
920 PRINT  @ 20,30;"LYNX ";L
;" YOU ";U;"  ";
930 PRINT  @ 20,40;"ANOTHER
 GAME? ";
940 LET A=GETN
950 IF A=89 OR A=121 THEN
 GOTO  LABEL NEXT.GAME
```

Machine Code for Beginners - Part 2.

Judging by what people have said it would seem that Machine Code for Beginners has been pitched about right. One or two points have been raised. Firstly some of you seem to think that you are going to pick machine code up in a few weeks and you feel that I should be explaining how to use it rather than just explaining concepts. Secondly some members feel that I am going a bit too slow. Consequently I have decided to pack quite a bit into this second part to get from the concepts of machine code to using it as soon as possible.

In this part I'm going to explain FLAGS and the STACK. Then I'm going to introduce a lot more opcodes and finally I'll give a couple of example programs with some exercises for you to try.

FLAGS

It was stated in part 1 that the purpose of the F register is to hold FLAGS. These flags indicate or flag the results of previous operations. There are seven flags in the F register but in order to keep things simple I am only going to deal with the two that I consider to be the most important. These are the CARRY flag and the ZERO flag.

I'm sure you can add 4 and 8 together in your head but do you remember the PROCESS that is performed. You probably learnt at your Infants School that 4 plus 8 is 2 carry 1. The point is that you can't put 12 in the units column it spills over into the tens column. In other words you 'carry one across'. The largest number that the A register may hold is 255 (decimal). Well what do you think happens when you add two numbers together into the A register and the result is bigger than 255? Absolutely correct! You 'carry one across' but in the Z80 there is no further column into which to carry the excess so it gets put into the CARRY flag. Not only is the carry flag used to indicate that the result is too big but also it is used if the result is negative. Suppose the A register contains 08 and you subtract 09 from it. In this case the carry flag would be set to indicate that the result is negative.

Having discussed the carry flag I'm sure you can guess what the ZERO flag does. If the result of a mathematical operation results in the A register becoming zero then the ZERO flag is set. It may sound a little theoretical but the flags are very useful since the results of operations can be used to direct the flow of a program.

The STACK

In the first part it was stated that there is something called a stack which works its way down from 9FF8H. Before we examine the Z80 stack lets look at a stack which you may be able to understand more easily. Suppose you ran a business and you had an arrangement with the Unions that if there are any redundancies they will be on a first in last out basis. In order to make sure that you get the order correct you keep a pile of postcards in your office. As a new worker joins you put his name on a new card and put it on the top of the pile. When the day comes for redundancies you take the cards off the top of the pile since you know that these people joined last. Obviously there are problems with such a system but it illustrates a Last In First Out or LIFO stack. It is this sort of stack which the Z80 uses. Purely for completenes I will mention the other type of stack which is First In First Out or FIFO. A queue is an example of this type of stack.

How is the Z80 stack organised? The Z80 has a special 16-bit register called the Stack Pointer or SP for short. This keeps track of where the last entry in the stack is.

A stack is started by loading the Stack Pointer with an address in ram where you want the stack to start. A stack works from high memory to low memory so it is logical to start the stack near the top of user ram. The opcode to load the SP is 31 and it should be followed by the 16-bit address of where you want the stack to start. The address as usual must be low byte first followed by the high byte. At memory locations 16B3-5 you will find the bytes 31 F8 9F. This starts the stack at location 9FF8H. Data in the form of 16-bit numbers may be PUSHED onto the stack or POPPED off the stack. The opcodes for pushing and popping the main register set are given below.

Table 1 -- PUSHES & POPS

REGISTER PAIR	PUSH	POP
AF	F5	F1
BC	C5	C1
DE	D5	D1
HL	E5	E1

Lets assume that we have started the stack at 9FF8H. What happens when we push data onto the stack? After we start the stack the situation would be as in the diagram below. The SP would contain 9FF8H and the ram would contain junk data.

```
ADDRESS    RAM      SP in the Z80
           +--+       +----+
9FF8       |j |       |9FF8|
9FF7       |u |       +----+
9FF6       |n |
9FF5       |k |
9FF4       |  |
9FF3       |d |
9FF2       |a |
9FF1       |t |
9FF0       |a |
```

If HL contained 1234H and we pushed HL onto the stack it would now look like this:-

```
ADDRESS    RAM      SP in the Z80
           +--+       +----+
9FF8       |j |       |9FF6|
9FF7       |12|       +----+
9FF6       |34|
9FF5       |k |
9FF4       |  |
9FF3       |d |
9FF2       |a |
9FF1       |t |
9FF0       |a |
```

The actual process that has taken place goes like this. First the SP is decremented. Secondly the high byte (ie H) is pushed to where the SP points (ie 9FF7). Then the SP is decremented again and now the low byte (ie L) is put on the stack. If DE contained 5678H and DE were pushed onto the stack it would now look like this:-

```
ADDRESS   RAM      SP in the Z80
          +--+     +----+
9FF8      |j |     |9FF4|
9FF7      |12|     +----+
9FF6      |34|
9FF5      |56|
9FF4      |78|
9FF3      |d |
9FF2      |a |
9FF1      |t |
9FF0      |a |
```

Now when we pop data off the stack the reverse process takes place. So if we pop DE off the stack it would look like this.

```
ADDRESS   RAM      SP in the Z80
          +--+     +----+
9FF8      |j |     |9FF6|
9FF7      |12|     +----+
9FF6      |34|
9FF5      |56|
9FF4      |78|
9FF3      |d |
9FF2      |a |
9FF1      |t |
9FF0      |a |
```

Naturally DE would end up containing 5678H.

Why is the stack so useful? You will soon find out that there are not enough registers in the Z80. The stack allows the programmer to save registers in a very efficient manner. Using the stack you can save 16-bit numbers by only using a single byte opcode. You can save the registers by loading them to ram but this would require three bytes per 16-bit number. As an example of this type of use you may care to look at memory locations 00C2H - 00CCH. Here you will find the following bytes:-

```
00C2    F5        Push AF onto the stack.
00C3    E5        Push HL.
00C4    D5        Push DE.
00C5    C5        Push BC.
00C6    CD E7 00  Call a routine at 00E7H.
00C9    C1        Pop BC off the stack.
00CA    D1        Pop DE.
00CB    E1        Pop HL.
00CC    F1        Pop AF.
```

Before a call is made to the routine at 00E7H all the main register set is saved on the stack. There are three things to note about this example. Firstly that it only takes four bytes to save the main register set - very efficient. Secondly note that the register pairs come off the stack in the reverse order from that in which they went on. It is most important to grasp this. Finally the PUSHing operation does not destroy the contents of the registers. In the example above the routine at 00E7H will be able to use the data in the registers as if no pushing had taken place.

CALL and RETURN

I have already made the point that it is the Program Counter (PC) which indicates from where the next opcode will be fetched. When a call opcode (CD) is executed it is followed by the address of where the call is to be made to. As you may have guessed the address is loaded into the PC and the Z80 simply carries on processing by picking its next byte up from the new address, but how does the Z80 know where to return to when it has finished the sub-routine?

After the call opcode (CD) has been picked up from memory the PC will be pointing to the first address byte. After the address bytes have been read from memory the PC will be pointing to the next opcode. It is this opcode which should be processed after the subroutine has finish-ed. Before the PC is changed to the subroutine address the contents of the PC are pushed onto the stack. So the return address is held on the stack. Then the new PC is set up. The subroutine is then executed and normally the last instruction in the subroutine is a RETURN opcode (C9). This opcode pops the return address off the stack and puts it into the PC. To illustrate this process lets look at the code above in more detail.

```
ADDRESS  OPCODE     EXPLANATION
00C2     F5         Push AF onto the stack.
00C3     E5         Push HL.
00C4     D5         Push DE.
00C5     C5         Push BC.
00C6     CD E7 00   Call a routine at 00E7H.
00C9     C1         Pop BC off the stack.
00CA     D1         Pop DE.
00CB     E1         Pop HL.
00CC     F1         Pop AF.
```

If the stack pointer were 9FF8H at 00C2H then after the opcode at 00C5 has been executed the register and stack would look like this:-

```
ADDRESS   RAM        SP        PC
          +----+     +----+    +----+
9FF8      |    |     |9FF0|    |00C6|
9FF7      | 'A'|     +----+    +----+
9FF6      | 'F'|
9FF5      | 'H'|
9FF4      | 'L'|
9FF3      | 'D'|
9FF2      | 'E'|
9FF1      | 'B'|
9FF0      | 'C'|
9FEF      |    |
9FEE      |    |
9FED      |    |
```

What data is actually on the stack doesn't matter for the explanation so I have just shown where the data from each register would be. The 'A' means that this is the data from register A and so on. Now the PC points to the CD opcode so the Z80 picks up the byte and increments the PC to point to the E7 address byte. When the Z80 picks up the CD opcode it knows that it needs the next two bytes to form an address. So it fetches the E7 byte and increments the PC. Finally it picks up the 00 byte and increments the PC. It now has the address and the PC now points to the C1 byte. This is where processing is to be continued from after the call has been made to 00E7H. So before the PC is changed to 00E7H the contents of the PC are pushed onto the stack. The PC is then changed to 00E7H. The registers and stack would now look like this:-

```
ADDRESS    RAM        SP        PC
          +----+    +----+    +----+
9FF8      |    |    |9FEE|    |00E7|
9FF7      | 'A'|    +----+    +----+
9FF6      | 'F'|
9FF5      | 'H'|
9FF4      | 'L'|
9FF3      | 'D'|
9FF2      | 'E'|
9FF1      | 'B'|
9FF0      | 'C'|
9FEF      | C9 |
9FEE      | 00 |
9FED      |    |
```

The Z80 now picks up the next opcode from 00E7H because that is where the PC points to. The subroutine will end with a RETURN instruction. This instruction will pop an address off the stack and put it into the PC. In this case it will remove the value 00C9H. The registers and ram would now look like this:-

```
ADDRESS    RAM        SP        PC
          +----+    +----+    +----+
9FF8      |    |    |9FF0|    |00C9|
9FF7      | 'A'|    +----+    +----+
9FF6      | 'F'|
9FF5      | 'H'|
9FF4      | 'L'|
9FF3      | 'D'|
9FF2      | 'E'|
9FF1      | 'B'|
9FF0      | 'C'|
9FEF      | C9 |
9FEE      | 00 |
9FED      |    |
```

One very important point is that subroutines should not put data on the stack and then not remove it. Suppose the subroutine pushed 0000 on the stack and then didn't remove it. When the RETURN instruction were executed it would remove 0000 and not the 00C9H value and consequently control of the program would go wrong.

You have already seen the call opcode it is CD. There are several others. They are CONDITIONAL CALLS. They are only executed if some condition is satisfied. The conditions are all connected with the flags we met earlier. The call opcodes are given in the following table:-

Table 2 -- CALL instructions

CD llhh	Unconditional call to hhll.
DC llhh	Call to hhll if carry flag is set
D4 llhh	Call to hhll if carry is not set
C4 llhh	Call to hhll if zero is not set
CC llhh	Call to hhll if zero is set

As well as the unconditional return opcode (C9) which you have already used there are conditional return opcodes. I have listed them in the next table.

Table 3 -- Return opcodes

C9	Unconditional return
D8	Return if carry flag is set
D0	Return if carry flag is not set
C0	Return if zero is not set
C8	Return if zero is set

JUMPS

The flow of a machine code program can be changed by JUMP opcodes. These opcodes jump to a specified memory location. Below is a table of some of the JUMP opcodes.

Table 4 -- JUMP opcodes

C3 llhh	Unconditional jump to hhll
DA llhh	Jump to hhll if carry
D2 llhh	Jump to hhll if no carry
CA llhh	Jump to hhll if zero
C2 llhh	Jump to hhll if not zero

You have already seen that it is possible to load registers with data from memory. In the table below I have listed some of the available 8-bit load instructions.

Table 5 -- 8-BIT LOADS

		SOURCE								
		A	B	C	D	E	H	L	(HL)	nn
DESTINATION	A	7F	78	79	7A	7B	7C	7D	7E	3E nn
	B	47	40	41	42	43	44	45	46	06 nn
	C	4F	48	49	4A	4B	4C	4D	4E	0E nn
	D	57	50	51	52	53	54	55	56	16 nn
	E	5F	58	59	5A	5B	5C	5D	5E	1E nn
	H	67	60	61	62	63	64	65	66	26 nn
	L	6F	68	69	6A	6B	6C	6D	6E	2E nn
	(HL)	77	70	71	72	73	74	75		36 nn

There are a few more 8-bit loads which I have listed below.
To load A with what BC points to use opcode 0A.
To load A with what DE points to use opcode 1A.
To load A from a location in memory use 3A llhh. Where the ll hh is the address with low byte followed by the high byte.
To load A to where BC points use opcode 02.
To load A to where DE points use opcode 12.
To load A to a location in memory use 32 hhll. Where hhll is the address with the low byte first.

A few examples should explain how to use the table. I'll start with one you have already met. To load the B register with a value you use 06 nn where nn is the value. The DESTINATION is the B register and the source is nn (ie data) so you can read 06 nn out of the table.

The HL in brackets (HL) means 'what HL is pointing to'. You have already met this in the first part of the series. So to load the C register with what HL is pointing to simply use the opcode 4E.

To load the D register with the E register use opcode 53.

To load what HL is pointing to with the contents of the D register use opcode 72.

16-BIT LOADS

You have already met some of these instructions. Here is a list of the more commonly used 16-bit loads.
To load HL with hhll use 21 ll hh.
To load DE with hhll use 11 ll hh.
To load BC with hhll use 01 ll hh.
To load HL with the data from location hhll use 2A ll hh
To load DE with the data at location hhll use ED5B ll hh.
To load BC with the data at location hhll use ED4B ll hh.
To load HL to location hhll use 22 ll hh.
To load DE to location hhll use ED53 ll hh.
To load BC to location hhll use ED43 ll hh.

Eight-bit addition is done by adding another register or data into the A register. The following table gives the opcodes.

Table 6 8-BIT ADDITION

A	B	C	D	E	H	L	(HL)	nn
87	80	81	82	83	84	85	86	C6 nn

The registers may also be INCREMENTED and DECREMENTED by one. The following table gives the opcodes for these operations.

Table 7 -- 8-BIT INCREMENTS and DECREMENTS

	A	B	C	D	E	H	L	(HL)
INCREMENT	3C	04	0C	14	1C	24	2C	34
DECREMENT	3D	05	0D	15	1D	25	2D	35

16-BIT Arithmetic

The following opcodes may be used to perform 16-bit arithmetic on register pairs.
To add BC into HL use 09
To add DE into HL use 19
To add HL into HL ise 29
To increment HL by one use 23
To increment DE by one use 13
To increment BC by one use 03
To decrement HL by one use 2B
To decrement DE by one use 1B
To decrement BC by one use 0B

I have just given you a lot of new opcodes. Now you will find out how to use some of them.

SUMS on the Z80

Hexadecimal addition isn't nearly as bad as it might seem. If you remember the process you go through when you add decimal numbers together then hex shouldn't be too much of a problem. When you add in decimal you add the units column first then the tens column and so on. With hex you do exactly the same. Consider the following:-

```
   AE    (note both numbers are hex)
 + 24
 ----

 ----
```

To add the units column you take the E and the 4, remember that E is 14 (decimal), and you should get 18 (decimal). If you can't remember the hex numbers from 0 to F then I suggest you get your copy of Issue 2 and you will find them on page 5. Now 18 (decimal) can't be expressed as a single hex digit because it is 12H. So you write in the 2 and carry 1 across. Now you take the A and the 2 which I make 12(decimal) and add in the carry to get 13 (decimal). Now 13(decimal) is DH. So the final sum looks like this:-

```
 AE
+24
---
 D2
---
  1  (the carry across)
```

Now lets try a simple program to do a little addition. As before I suggest you load the programs at memory location 9000H. The results of all 8-bit calculations end up in the A register. So if you want to add two numbers together you put one in the A register and then add the other number into the A register.

To add 78H to 23H then the program could look like this:-
1. Load the A register with 78H.
2. Load the B register with 23H.
3. Add the B register into the A register.
4. Load the A register into ram so that the result may be examined.
5. Return to the monitor.

You should be able to pick the opcodes out of the tables to form this program. I make it:-

```
1. 3E 78      Load A with 78H
2. 06 23      Load B with 23H.
```

```
 3. 80          Add B into A.
 4. 32 09 90    Load A to location 9009H
 5. C9          Return to monitor.
```

How did I know where to put the result? The answer is quite simple because as long as it is loaded somewhere into available user ram the program will work. If the program is loaded at 9000H the C9 opcode will be at 9008H so it seemed sensible to load the the result into the next available byte which is 9009H. Now after you have loaded the program run it by typing G9000 [RETURN] and then look at the result in location 9009H. Try changing the values that you load into A and B and rerun the program. See if you can make sense of the results. Try changing the program so that it INCREMENTS or DECREMENTS a register and then saves the result so that you can see the effect.

Now we will have a look at a routine to blink an asterisk on the screen. The logic to do this would look something like this:-

```
1. Display an asterisk
2. Wait a while
3. Delete the asterisk.
4. Wait a while.
5. Go back to 1.
```

The only problem with the logic is that once it started it would never stop! You would have to pull the plug out of your Lynx to get control again. Consequently I suggest that we blink the asterisk (say) 16 times. The logic would now look like this:-

```
1. Set the counter
2. Display the asterisk
3. Wait a while.
4. Delete the asterisk
5. Wait a while
6. Decrement the counter
7. Test if zero
8. Goto 2 if not zero
9. Return.
```

You will have noted that the 'Wait a while' occurs twice. This makes it an ideal candidate for a subroutine. To 'wait a while' all you need to do is to get the Z80 to do some dummy processing in a loop. I suggest that we get it to count down from a large number. The logic for the subroutine looks like this:-

```
1. Set HL to a large number
2. Do some dummy work
3. Decrement HL
4. Test if it is zero
5. Goto 2 if not zero
6. Return
```

I suggest that the dummy work we give the routine to do is to PUSH a register pair onto the stack and then POP it off again. The routine now looks like this:-

```
1. 21 00 80  Load HL with 8000H
2. C5 C1     Push BC and pop BC
3. 2B        Dec HL
4. 7C        Load H into A
   85        Add L into A
5. C2 11hh   Jump if not zero
6. C9        Return
```

First of all note that the operation of adding L into the A register will set the ZERO flag if both H and L contain zero. Secondly note that at this stage we do not know where the jump (line 5) is to be made to so for the time being we will leave it blank. Now lets consider the main routine. We can display an asterisk by loading the A register with the ASCII code for an asterisk and calling the character output routine. We did that in Part 1. To delete it we can use the same routine to output a backspace. The main routine now looks something like this:-

```
1. 21 10 00  Load HL with 16
2. 3E 2A     Load A with ASCII for *
   CD A4 06  Call character output
3. CD 11 hh  Call delay routine
4. 3E 08     Load A with Backspace
             See VDU codes on page 62
             of the user manual.
   CD A4 06  Call character output
5. CD 11 hh  Call delay routine
6. 2B        Decrement HL
7. 7C        Load H into A
   85        Add L into A
8. C2 11 hh  Go to 2 if not zero
9. C9        Return
```

Now we still have a few addresses missing. Furthermore both the main routine and the delay subroutine use the HL register pair as a counter. Consequently when the delay subroutine is called it will destroy the contents of the HL register pair which is used as a counter in the main routine. This can be avoided by pushing the contents of HL onto the stack in the main routine to preserve the value.

If you put the routine into memory starting at 9000H and follow it with the delay routine you will then be able to work out the missing addresses. It should look like this:-

```
ADDRESS OPCODES   EXPLANATION
9000    21 10 00  Load HL with 16
9003    E5        Push HL to protect HL
9004    3E 2A     Load A with ASCII for *
9006    CD A4 06  Call character output
9009    CD 1C 90  Call delay routine at 901C
900C    3E 08     Load A with Backspace
900E    CD A4 06  Call character output
9011    CD 1C 90  Call delay routine at 901C
9014    E1        Pop HL to restore HL
9015    2B        Decrement HL
9016    7C        Load H into A
9017    85        Add L into A
9018    C2 03 90  Jump if not zero to 9003
901B    C9        Return
Now for the delay routine.
901C    21 00 80  Load HL with 8000H
901F    C5 C1     Push BC and pop BC
9021    2B        Dec HL
9022    7C        Load H into A
9023    85        Add L into A
9024    C2 1F 90  Jump if not zero to 901F
9027    C9        Return
```

For those of you that want to pull your hair out try to re-write the program to work somewhere else, say at 8800H. Secondly try to make the asterisk blink MUCH slower say at once every two seconds. Have fun!

DRIVING A PARALLEL PRINTER

In the previous issue a design to give you parallel ports on your Lynx was published. This article describes how to use the circuit to drive a printer with a Centronics interface.

The Centronics interface is composed of 36 signal lines. Eight of them are the data lines and the rest are either control signals, ground return lines or not used.

Data may be sent via a Centronics interface at several thousand characters per second whereas a printer is normally capable of printing at about 80-120 characters per second. Unless there is some form of control signal a lot of data is going to get lost or, worse still, the printer will be damaged. When the printer is busy it generates a BUSY signal which it sends to tell the computer not to send data. So the first thing a driver routine must do is to examine the BUSY signal to see if the printer is readv to receive characters. When the printer is ready the computer sends a character to the printer and then sends a signal to the printer to say that valid data is on the data lines. This signal is called STROBE. When the printer sees that the STROBE signal is active it will read the data and generate a BUSY signal to prevent further data being transmitted. This procedure is repeated for all characters.

There are other signals on the Centronics interface such as 'error' and 'out of paper' but STROBE and BUSY are normally sufficient to be able drive a printer.

The parallel ports need to be initialised before printing takes place. How you initialise will depend upon how you configure the wires onto the various port pins. I chose to use port B to output the data and port C (upper) for the input control signal (ie BUSY) and port C (lower) for the output control signal (ie STROBE). The BUSY signal is on pin 7 and the STROBE signal is on pin 0.
The following lines can be used to drive a printer.

```
10 CODE 3E 88 D3 4E C9
20 CODE F5 DB 4D CB 7F 20 FA F1 F5 FE 1F 20 02 3E 0D
D3 4A 00 AF D3 4C 00 CB C7 D3 4C F1 C9
```

You use CALL LCTN(10) to initialise the interface and DPOKE &6202,LCTN(20) to set up the printer driver routine. You may now use LINK ON/OFF, LPRINT and LLIST.

The following is the assembler for the two routines.

```
8000           0010        ORG  £8000
8000 000D      0020 CR     EQU  £D
8000 004A      0050 WPORTB EQU  £4A ;Port B write
8000 004C      0070 WPORTC EQU  £4C ;Port C write
8000 004D      0080 RPORTC EQU  £4D ;Port C read
8000 004E      0090 WCTL   EQU  £4E ;Control port
               0110 ;INITIALISATION ROUTINE
8000 3E88      0150        LD   A,£88
8002 D34E      0160        OUT  (WCTL),A
8004 C9        0170        RET
               0190 ;PRINTER DRIVER
8005 F5        0200        PUSH AF ;Save character
8006 DB4D      0210 LOOP   IN   A,(RPORTC) ;Read BUSY
8008 CB7F      0220        BIT  7,A ;Test BUSY
800A 20FA      0230        JR   NZ,LOOP ;Loop until ok
800C F1        0240        POP  AF ;Return character
800D F5        0250        PUSH AF ;Save character
               0260 ;Is it 1FH ;ie should be CR
800E FE1F      0270        CP   £1F
8010 2002      0280        JR   NZ,CHROUT
8012 3E0D      0290        LD   A,CR
8014 D34A      0300 CHROUT OUT  (WPORTB),A
               0310 ;Now strobe
8016 00        0320        NOP  ;Let data settle
8017 AF        0330        XOR  A
8018 D34C      0340        OUT  (WPORTC),A ;STROBE low
801A 00        0350        NOP  ;let STROBE settle
801B CBC7      0360        SET  0,A ;Reset STROBE
801D D34C      0370        OUT  (WPORTC),A
801F F1        0390        POP  AF ;Return character
8020 C9        0400        RET
```

One final point. Unfortunately there was a mistake on the circuit diagram published in Issue 2. The INT line going into IC2b should have been M1 (pin 32 on the interface connector).

A/D INPUT

The Lynx has an analogue to digital input on pin 2 of the lightpen socket. By applying a voltage in the range 0-3.4 volts you can program the Lynx to give you a digital reading in the range 0-63. Admittedly the resolution is not very good but its better than nothing. The port is activated by changing register 12 of the 6845 CRTC to 20H. Then you apply a comparison voltage to the A/D output port (84H). Finally you compare the input and output voltages by testing bit 0 of port 80H.

The whole process can be done in BASIC but it is possible to crash the Lynx by escaping from the routine while it is running so it is best done in a small machine code routine. The routine applies a 'rising ramp' test voltage to the output port and then detects when bit 0 of port 80H changes. When it does the output voltage may be considered equal to the input voltage. The routine may be used in machine code or incorporated into a CODE line for use in BASIC. To read the port use CALL LCTN(10). The result is returned in HL and so may be used in BASIC.

```
10 CODE 3E 0C D3 86 3E 20 D3 87 21 00 00 01 80 0B 7D  D3  84
11  E8 03 1B 7A B3 20 FB ED 78 CB 47 28 05 2C CB 75 28 EA AF
D3 87 C9
```

The assembler for the routine.

```
8000           0010        ORG  £8000
               0015 ;Activate the A/D input port
8000 3E0C      0020        LD   A,12
8002 D386      0030        OUT  (£86),A
8004 3E20      0040        LD   A,£20
8006 D387      0050        OUT  (£87),A
8008 210000    0060        LD   HL,0
800B 01800B    0070        LD   BC,£0B80
800E 7D        0080 LOOP   LD   A,L
               0085 ;Set up comparison voltage
800F D384      0090        OUT  (£84),A
8011 11E803    0100        LD DE,1000 ;Delay loop
8014 1B        0110 DLOOP  DEC  DE
8015 7A        0120        LD   A,D
8016 B3        0130        OR   E
8017 20FB      0140        JR   NZ,DLOOP
8019 ED78      0150        IN   A,(C); Read port 80
801B CB47      0160        BIT  0,A ;Test bit 0
801D 2805      0170        JR   Z,FOUND
801F 2C        0180        INC  L;Increment test value
8020 CB75      0190        BIT  6,L
8022 28EA      0200        JR   Z,LOOP
8024 AF        0210 FOUND  XOR  A ;Deactivate port
8025 D387      0220        OUT  (£87),A
8027 C9        0230        RET
```

Thanks go to David Barnes for information about the A/D input port.

Using the BREAK Key

I wish I had a pound (Sterling not weight!) for every time someone has told me that the BREAK key is not connected to anything because it is connected to the maskable inter-rupt pin of the Z80. The significance of this fact may be lost on some of you but the results may not. I have managed to set the BREAK key up as a clear-screen key and I've found it quite useful. I see no reason why it can't be set up for other uses.

Before I explain how its done I am going to attempt the impossible and explain interrupts in a few sentences. Interrupts fall into the category of Machine Code for Experts so if you have just started learning machine code don't be too surprise if you don't understand them.

You may be interrupted during the day by a phone call. You stop what you are doing (even if you are doing nothing) and answer the call. When its finished you go back to whatever you were doing before the call. You may decide that you don't want to be interrupted and so you may 'disable' the phone by taking the hand-set off the phone. That, in a nut-shell, is what interrupts are all about. .

The Z80 can process interrupts. It can be doing one thing and then in comes an interrupt. It stops what it is doing, goes and answers the interrupt and then comes back to what it was doing before the interrupt came in. Its all very fine in theory but in practice its a lot more complicated. To start off with there are two types of interrupt Maskable and Non-Maskable. The Z80 can be set up to ignore Maskable Interrupts but it can't ignore Non-Maskable interrupts. Furthermore there are three modes of maskable interrupts (ie three different ways in which the Z80 can respond to an interrupt).

On top of all this is the fact that interrupts may be nested. This means that while the Z80 is responding to one interrupt another interrupt comes in. To explain this concept in human terms imagine that you are in your office and have been interrupted by a phone call. While you are still talking your other phone rings. Chances are you will stop the first conversation, ask the second caller to hold the line because you are already on the phone, and then you go back to the first caller.

I hope that the concept of interrupts is clear even if you don't understand the mechanics. I will no doubt cover interrupts in greater depth in some future articles.

When the LYNX powers up the very first thing it does is to disable interrupts. There is a good reason for this. The LYNX's hardware is set up to generate maskable interr-upts. In fact it is the cursor pin of the 6845 that does this. Maskable interrupts are generated at the rate of 50 per second. If interrupts were not disabled the LYNX would receive its first interrupt while it was setting itself up with unpredictable results.

I have found the my LYNX has a power-on interrupt mode of 1. I don't know if this is normal because I can't find any instruction in the BASIC EPROMS to set this mode and it doesn't mention a default interrupt mode in the Z80 Technical Manual. This interrupt mode is, in fact, the one required. In this mode the Z80 responds to an interrupt by calling location 0038H. This has a jump to location 6297H which in turn jumps to the error routine. The ram vector (at 6297H) may be changed to point to your own interrupt routine but there are several problems to be overcome first.

The first is that as soon as you enable interrupts you will be greeted with the NOT YET IMPLEMENTED error message. You can prove this quite easily. Enter the following lines:-

```
        10 CODE FB C9 [RETURN]
        CALL LCTN(10) [RETURN]
```

Line 10 contains a subroutine (one opcode and a return!) which will enable interrupts. As soon as interrupts are enabled one comes in from the 6845 and up comes the message. The 6845 cursor pin can be disabled by changing the contents of register 14 on the 6845. This can be done with the following two lines:-

```
        OUT &86,14  [RETURN]
        OUT &87,&3F [RETURN]
```

If you now CALL LCTN(10) you will find that you don't get the error message until you press the BREAK key. Under normal circumstances you would then be able to point the ram vector at 6297H to your own interrupt routine and that would be that.

Unfortunately with the LYNX its not quite that simple. To begin with you MUST prevent interrupts being generated while video ram is being accessed. This is necessary because the Z80 stack is used during interrupt processing and is unavailable during video updates. The second problem concerns the status of address lines A6 and A7. It states in the first edition of LYNX USER that A7 must be low or A6 must be high. I think that it is impossible to ensure that one of these conditions exists when an interrupt is generated. The third problem is that under exceptional circumstances it is possible for the LYNX to generate a non-maskable interrupt while it is responding to a maskable interrupt!

I think that I have solutions to each of these problems. The non-maskable interrupt can be 'disabled' by setting up the ram vector at 6294H to be a RETN instruction. When the NMI comes in it is simply ignored by the software. To prevent interrupts during screen access is quite straight-forward. Simply change the VDU driver address at 6200H to point to a new driver routine. The new routine should disable interrupts, access the VDU ram as normal and then enable interrupts again.

I have no guaranteed solution to the A6/A7 problem. I have, however, found something that seems to work. The A6/A7 business is a problem because when an interrupt is generated the IO request signal becomes active and this coupled with a correct address can enable one of several ports. In particu-lar ports 80H and 84H can cause problems. I have found that by writing 00 to these ports after an interrupt has been generated solves this problem. I have put my clear-screen routine into an EPROM and so far I've had no problems with it.

So now to the routine. The first thing to note is that not only is it a routine but it is also an initialisat-ion routine as well. In the form given it MUST be the first line in BASIC. Don't make the mistake of entering a line of BASIC in front of this line.

```
10 CODE ED 56 E5 F5 AF D3 80 D3 84 21 ED 45 22 94 62 21 54
69 22 98 62 21 A4 06 22 00  62 3E 0E D3 86 3E 3F D3 87 3E 04
CF 21 84 69 22 00 62 F1 E1 FB C9 F3 CD A4 06 FB C9
```

To initialize the routine simply enter:-

```
        CALL LCTN(10) [RETURN]
```

Now if what I have been saying about the Z80 being able to handle two functions is true then you should be able to set the LYNX to do one thing and clear the screen at the same time. Try setting the LYNX up to print lines of text and pressing the BREAK key.

Although it has been my work and perserverance that has unearthed how to use the BREAK key I wish to acknowledge the considerable help I have received from others. Thanks go to Chris Cytera, Francis Lovering and Evan Shuttock for pieces of information which helped complete the jig-saw puzzle.

```
                0010 ;This routine MUST be located
                0020 ;in a CODE line which
                0030 ;MUST be the first line in BASIC
6954            0040        ORG  £694D+7 ;Byte after CODE
6954 06A4       0050 NDRIVE EQU  £06A4 ;Normal vdu driver
6954 6200       0060 NVDU   EQU  £6200 ;Location of vdu vector
6954 6294       0070 NMIRAM EQU  £6294 ;NMI ram jump
6954 6297       0080 RSTRAM EQU  £6297 ;Interrupt jump


6954 ED56       0110 START  IM   1 ;Set interrupt mode
6956 E5         0120        PUSH HL ;Save registers used by
6957 F5         0130        PUSH AF ;this routine
                0140 ;Reset ports £80 and £84 which may have
                0150 ;been corrupted by the interrupt
6958 AF         0160        XOR  A
6959 D380       0170        OUT  (£80),A
695B D384       0180        OUT  (£84),A
                0190 ;'Disable' the non-maskable interrupt
                0200 ;by setting up an immediate return
695D 21ED45     0210        LD   HL,£45ED ;For 'RETN'
6960 229462     0220        LD   (NMIRAM),HL
                0230 ;Set up ram jump to the interrupt routine
6963 215469     0240        LD   HL,START
6966 229862     0250        LD   (RSTRAM+1),HL
                0260 ;Set up the normal vdu driver
                0270 ;This is necessary to prevent a second
                0280 ;interrupt ocurring after the screen
                0290 ;has been cleared
6969 21A406     0300        LD   HL,NDRIVE
696C 220062     0310        LD   (NVDU),HL
                0320 ;Disable the 6845 cursor interrupt
696F 3E0E       0330        LD   A,14
6971 D386       0340        OUT  (£86),A
6973 3E3F       0350        LD   A,£3F
6975 D387       0360        OUT  (£87),A
                0370 ;Finally! Clear the screen
6977 3E04       0380        LD   A,4
6979 CF         0390        RST  8
                0400 ;Set up the vdu driver so that
                0410 ;interrupts can't be generated during
                0420 ;output to the screen
697A 218469     0430        LD   HL,MYVDU
697D 220062     0440        LD   (NVDU),HL
6980 F1         0450        POP  AF ;Restore registers
6981 E1         0460        POP  HL
6982 FB         0470        EI   ;Enable interrupts
                0480 ;RETI is not necessary since nested
                0490 ;interrupts are not catered for
6983 C9         0500        RET

                0520 ;A vdu driver which
                0530 ;will prevent interrupt during
                0540 ;vdu access
6984 F3         0550 MYVDU  DI
6985 CDA406     0560        CALL NDRIVE
6988 FB         0570        EI
6989 C9         0580        RET
```

STRUCTURED PROGRAMMING by Chris Cytera

This article is intended as a guideline for producing clearly written structured programs using the LYNX. It is not intended as the last word on structured programming, as whole volumes have been written on the subject. It does, however, contain what are intended to be helpful DOs and DON'Ts, as well as some hints on how to shed some of those bad habits picked up from using more primative versions of BASIC than that on the LYNX.

There are two distinct steps to writing a program. The first stage is to design the logic with which the program is to solve the problem, move space invaders around the screen, or whatever it is to do. This is called the ALGORITHM and you are most likely to have met this before in the form of a flowchart. However, flowcharts have their disadvantages as I shall point out later.

The second step is to translate this algorithm into the particular language that you are using, so that the computer can understand it. This is known as CODING. This should in theory be the easy step, because all the problem-solving methods that the program uses should have been worked out by this stage. It is up to you to ensure that this is so in practice, by defining your algorithm as clearly and as precisely as possible.

It is when the distinction between the two steps is blurred that the trouble begins. A bad way to write a program is to switch the computer on, quickly 'knock-up' a kind of first draft, and then spend the rest of the time adding patches and 'bodges' in order to get it to work. The result is invariably a jumbled mess riddled with GOTO statements. Programs like this tend to end up with statements such as 200 GOTO 930 and line 930 GOTO 2000 and at 2000 GOTO 600 and so on. All this because certain lines were missed out and the programmer redirected the program flow somewhere else. Programs like this are known as 'spaghetti programs' in the trade.

So how should a program be written? As I have already said, the algorithm should be written first. This should be precisely defined and totally unambiguous. It is not enough to think you know what the program is to do - write it down. This will help to clarify your thoughts. If possible, get somebody else to read your algorithm - if they cannot make head nor tail of it then you have little hope of turning it into a program. The algorithm should be broken down into separate modules, each of which performs a certain function. These will correspond to the PROCedures of LYNX BASIC when you come to code the program. The ability to break a task down in this way is the key to structured programming and it has numerous advantages:

1. The modules, which should be independent of each other, can be tested individually before they are put together to make the final program. This is not possible with 'spaghetti' programs which have GOTOs leaping in and out of sections of code, making them interdependent.

2. If a fault does occur, then its identification and location is made easier because each procedure is associat -ed with its own distinct function.

3. The procedure can be 'lifted out' and added to a library, for use in other programs.

4. By using meaningful PROCedure names, the legibil -ity of the program is greatly increased.

5. By keeping notes on the function of each proce -dure, the program can be returned to several months later

and the same train of thought picked up again fairly easily. Keep a list of the variables you are using and their function. Try to use meaningful variables where possible, for example D for Day and R for Room. Sadly, the LYNX's single letter variables greatly restrict the practice; furthermore, some of the lowercase letters look too much like their uppercase counterparts for comfort, the worst culprit being the letter K. Long variable names would be a welcome addition to the extended BASIC ROM, if indeed their implementaion is possible; the speed and memory penalties involved would be well worthwhile.

6. If the program requires drastic modification at a later date, perhaps due to a change in specification, then all that is required is to add more procedures to carry out the extra functions, and to insert procedure calls in the appropriate places. An unstructured program will inevitably require the addition of yet more GOTO statements to direct the program flow to the new code and back again, plunging it further into disarray.

Unfortunately LYNX BASIC does not at pressent support variables local to procedures; that is, variables created within a procedure which do not affect the values of outside variables with the same name. This makes recursion (PROCs calling themselves) on the LYNX more difficult than it would otherwise be.

It is the criticism of GOTOs which brings me onto flowcharts. Although they are better than nothing as a means of representing algorithms, they do encourage an unstructur-ed approach by having lines jumping around between various boxes. They are also difficult to edit. It is for these reasons that they are not very highly regarded by many professional programmers.

It is better, in many people's opinion ,to write down the algorithm in words. If you like, you can call it your own computer language which you invent as you go along to suit your purpose. For example, consider the writing of a simple bat-and-ball type game. The algorithm for such a program would look something like this:

```
Initialise variables
Set up screen
Repeat
  Repeat
    Serve ball
    Do while ball is in court:
      Scan keyboard
      Move bats
      Move ball
      If ball hits obstacle then change
      direction of ball
    End while
    Adjust score
  Until somebody wins
Until players have had enough!
```

This type of approach is much clearer than a mass of boxes joined together by a tangle of lines with arrows pointing in all directions. It also translates readily into the WHILE...WEND; REPEAT .. UNTIL and PROC facilities provided by LYNX BASIC. For example, the 'Move bats' could become a PROC BATS. This procedure will do the necessary checking to see if the bats have moved off the screen etc. as well as actually drawing the bats.

Although LYNX BASIC is not properly structured like PASCAL for example, the above-mentioned features and the ELSE clause help matters considerably. If you can imagine what BASIC would be like without FOR ... NEXT loops then these facilities each add as much sophistication again. Sadly missing are long variable names; local variables; CASE statements and BEGIN ... END blocks. Hopefully these will be incorporated in the forthcoming extended BASIC in prefer-ence to silly gimmicks such as pre-programmed sounds.

I will now cover each structure provided by Lynx BASIC in turn.

PROCEDURES

These have been provided in other high-level langu-ages since they were introduced, but only one other version of BASIC apart from that on the Lynx supports them. Procedures give us a way of naming a set of statements and executing them with one statement. This is similar to GOSUB but more powerful. It might be best to forget that GOSUB exists at all (except for the computed variety) as it is just an inferior procedure call. Incidently PROC executes more quickly than both GOSUB and GOSUB LABEL.

A good program structure to aim for would be a 'master' section of the program that just contains procedure calls surrounded by FOR ... NEXT, REPEAT ... UNTIL and WHILE ... WEND statements. Another feature that adds to the power of procedures is parameter passing. This gives us the ability to make procedures general, so that they will produce a different result with different parameters. This also enables us to use procedures as definable functions, returning values; one advantage of not having local vari-ables. Oddly and annoyingly, string arguments cannot be passed to procedures without a separate assignment state-ment.

REPEAT ... UNTIL and WHILE ... WEND

Both of these loops are substitutes for the FOR ... NEXT loop when it is not known how many times the loop will execute. The conditional loops are much clearer that just using GOTO statements alone. The WHILE loop is slightly different because the condition is specified at the beginn-ing. This means that the loop is skipped altogether if the condition is false initially, whereas the REPEAT loop always executes at least once.

Here is an example of a program that uses these two loops together - it finds highest common factors by means of Euclid's algorithm.

```
100 INPUT "First factor ";N
110 INPUT "Second factor ";M
120 REPEAT
130   WHILE N>M
140     N=N-M
150   WEND
160   WHILE M>N
170     M=M-N
180   WEND
190 UNTIL N=M
200 PRINT "H.C.F. = ";N
```

SUMMARY

A problem should be reduced into sections, each section worked on separately and made into a procedure. Each procedure may use other procedures.

The master program should be short enough to grasp as a whole - about 40 lines or so. Keep a list of variables and procedures used.

Beware of using too many GOTOs. This is not just an ideological whim - it is accepted practice in professional circles. If you must use GOTO, for goodness sake use a LABEL as it makes things much clearer.

LYNX PROGRAMMERS

MACHINE CODE REACHES THE PARTS THAT BASIC CANNOT FOR EXAMPLE YOU CAN:–

SPEED UP THE SCREEN
MAKE BASIC PROGRAMS FASTER AND SMALLER WITH CODE LINES
USE ALTERNATE GREEN FOR PROGRAMS OR DATA
ADD TO CASSETTE HANDLING
ADD COMMANDS TO THE EDITOR

It need not be difficult – the LYNX has excellent facilities for incorporating M.C. in BASIC programs – and it may be that 30 or 50 bytes can make the world of difference. You may have already tried "hand" assembling using the monitor, but this is painfully slow.

"CODER" an instant assembler and disassembler for CODE lines, RAM and ROM makes it easy. 20 single key commands include insert and replace (in assembly mnemonics) delete, change an operand, save, test and many more. Operands can be entered in hex, decimal, binary or as expressions. Any M.C. in CODE lines can be relocated, and you can slip into monitor and back to CODER at will.

The 11 page manual includes simple routines for those just learning, and plenty of information about the program. "CODER" is in basic with machine code routines (in CODE lines) used extensively. It leaves room for over 4K of code on a 48K LYNX and includes with notes

AN ON GOTO ROUTINE (look up table)
A BANK SWITCHED FAST BLOCK PRINT ROUTINE – this prints on the screen – block any height and width in under a tenth of a second!

"CODER" CAN BE YOURS FOR JUST £7.50

Cheques or P.O. to:

**FL SOFTWARE,
13 St. Ronans Avenue,
Southsea,
Hants, PO4 0QE
Tel: (0705) 828295**

HIGH SPEED MACHINE CODE GAMES COMING SOON

LYNX 48K SOFTWARE

DISASSEMBLER £4.75 (£4.25)
This basic utility program enables you to examine machine code programs in ROM or RAM in standard Z-80 mnemonics. Features:– Full disassembler or relative and absolute jumps, option of disassembly in decimal or hexedecimal notation.

LABYRINTH £4.75 (£4.25)
This is a top quality 3-D maze game made possible by the superb graphic capabilities of the LYNX. You have to think spatially to find your way through the many complex mazes in the shortest route. Do not despair, help is at hand if needed.

CHANCELLOR £4.75 (£4.25)
This program provides a realistic simulation of the UK economy, working from either one of two economic theories (which are fully explained). Dare you unleash your ideas upon the unsuspecting population? Can your policies work? Try to remain in office for the full term of ten years.

REVERSALS £4.75 (£4.25)
This is our version of the popular board game, an exciting, fast and highly addictive game of logic and skill, easy to learn yet hard to master. The program is written in machine code, so no more long waits between moves.

SPACE TREK £4.75 (£4.25)
Space – the final frontier. You are in command of the Star Ship Enterprise, and the Federation of Planets is in grave danger from invading forces. Your mission – to save the Universe by seeking out and destroying the evil Klingons.

THE WORM £5.95 (£5.35)
Our best yet! Guide Wilberforce the worm to the flowers which are his staple diet. The problem is that he will try to eat rocks, walls and himself all of which are fatal. He also moves at incredible speed.

Nilug members price in brackets – quote mem. No.

QUAZAR COMPUTING DEPT. N.
29 WESTERN RD., NEWICK, E. SUSSEX
overseas – add 10%
Other programs available send for details

*** SCROLLER ***

Scrolls the screen properly on the 48K LYNX. Will scroll any primary colour on listing, text entry etc. Works in TEXT mode. Supplied in a chip (2764 EPROM) complete with instructions.

£10.00

10% Discount for NILUG members.

LYNXMAN
270, Tithe Pit Shaw Lane,
Warlingham,
Surrey CR3 9AQ.
Or send SAE for details.

Send s.a.e. for FREE CATALOGUE

SOFTWARE (Quazar) Labyrinth, Chancellor, Reversals, Space Trek, Disassembler £4.75 e.a. The Worm £5.95; (Level 9) Colossal Adventure, Adventure Quest, Dungeon Adventure, Snowball £9.90 e.a. (Gem) Monster Mine, Golf, Sultan's Maze, Gempack IV £7.95 e.a. Discounts off total; 2 off = 50p; 3 off = £1; 4 off = £2. BOOKS "Lynx Computing" £6.95 HARDWARE parallel connector + computer reset £4.50 DUST COVERS £2.75 CASSETTE LEADS DIN to DIN + remote or DIN to 3 jacks £2.35

FULCRUM PRODUCTS
14 STEEP LANE, FINDON
W SUSSEX BN14 0UF

ZEN

EDITOR ● ASSEMBLER ● DEBUGGER

A COMPLETE PACKAGE FOR WRITING & DEBUGGING Z80 MACHINE CODE PROGRAMS ON THE LYNX **£22.50**
C/W MANUAL AND CASSETTE

From LAURIE SHIELDS SOFTWARE
151, LONGEDGE LANE,
WINGERWORTH,
CHESTERFIELD.
S42 6PR.

MICRO COMPUTER WORLD

Lynx Invaders	£9.90	Numerons	£9.90
Power Blaster	£9.99	Moonfall	£7.95
Hangman	£7.95	Dambusters	£6.95
Connect 4	£4.95	Sultans Maze	£7.95
3D Monster Craze	£9.99	Monster Mine	£7.95
Golf	£7.95	Gamespack 3	£7.95
Gempack 4	£7.95	Mined Out	£7.95
Joystick Interface	£14.95	Games by Lynx & Gem	

Parallel Printer Interface £49.95

Cheques/P.O. to Micro Computer World
All prices inc. VAT P & P free
For free descriptive catalogue write to:

**Micro Computer World
1 Lane Close,
London NW2 6QZ**