NILUG NEWS

NATIONAL INDEPENDENT LYNX USER GROUP

Magazine for **LYNX** Users

Volume 1. Issue 2.

NILUG NEWS

VOLUME 1. ISSUE 2.

CONTENTS

ANNUAL SUBSCRIPTIONS (SIX ISSUES)

U.K.	£9.00 per year.
OVERSEAS	£12.00 per year.

Send cheque, payable to NIULG, to :-
NILUG
53,KINGSWOOD AVENUE
SANDERSTEAD
SOUTH CROYDON
SURREY
CR2 9DQ

EDITORIAL

As many of you will know NILUG was launched at the Computer Fair. I was fortunate in being able to talk to many LYNX owners and find out what they expect from a user group. The demand for 'machine code for beginners' was enormous. I estimate that about 80% of the people I spoke to wanted it. Consequently a series is starting on Machine Code for Beginners. I hope to get a lot of feedback on this topic. This will allow me to find out what points you have difficulty with.

NILUG is growing but not fast enough to warrant an increase in the size of NILUG NEWS. This is unfortunate because I have so much copy that its hard to know what to leave out. I had planned to publish details on interfacing printers (both parallel and serial) as well as reading the A/D input port. These will have to wait. As you will see this issue is printed in a much smaller type face. This has enabled me to publish about 22 pages of copy on only 12 physical pages. Please keep the copy coming. Programs, tips, reviews etc will all be most welcome.

Several members have asked me to supply them with names and addresses of local LYNX users. I feel that it would be wrong to give out details without permission. If you want to contact local users I would be only too happy to publish your name and address in the next issue.

NILUG as the name implies was set up as a national user group. It has come as a pleasant surprise to receive several requests for membership from overseas. I am only too happy to oblige. The subscription is £12.00 per year. The difference is purely the extra cost of postage.

NILUG will be taking a stand at the PCW show September 28th - October 2nd. I hope to see some of you there.

TECHNICAL ENQUIRIES

Several people have written to me with technical enquiries. Although I don't guarantee to reply there are several things you can do to improve your chances.

1. SEND AN SAE. I'm not going to pay the postage and I don't think that its fair on other NILUG members for NILUG to pay. People sending an sae go to the top of the pile when it comes to receiving replies. The exception to this is if I consider that there may be an article for NILUG NEWS hiding in your letter.

2. ASK DIRECT QUESTIONS. If you say 'on my machine the left wiget is on the right' what do you expect from me? If, however, you say 'on my machine the left wiget is on the right DOES THIS MATTER?' I know exactly what your question is.

3. DON'T ASK ME TO EXPLAIN 'RELATIVITY'. If you ask questions which will require several pages of explanation you stand no chance of getting a reply. Don't,for example, ask me to explain 'The Z80','Output to the screen' etc. By all means ask about ONE aspect with which you may be having problems.

4. REPEAT YOUR QUESTIONS AT THE END. It is most helpful to have the questions repeated at the end of the letter. If I have to read through several pages to find your questions I'm very likely to miss them.

5. GIVE SOME BACKGROUND. In answering your letters I need to know what you can understand. If you are a beginner then I must reply in terms that you will understand. Try to rate yourself from 0 to 9 in the following three areas:-

1. Hardware. 2. Machine code / assembler. 3. Basic.

Be honest about it and there is no prize for the first 0/0/0 to come in. It also helps if you give some background to the problem. Why is it a problem to you? What have you tried? What happened?

6. ASK ABOUT ONE THING ONLY (MAYBE TWO). Its very easy to write 'can you help me with a modem, a printer interface, colour graphics .... etc'. If you must write about several items then don't expect me to answer all your questions.

7. THE ANSWER MAY BE COMING. It may be the case that so many people have asked your question that the answer will be published in the next issue of NILUG NEWS. Interfacing printers, for example, is on its way.

8. I AM NOT CAMPUTERS. I am NOT (I said NOT) Camputers. Make sure that you address your problems to the correct place. If your left wiget has fallen off your LYNX then contact your dealer or Camputers, NOT me.

9. REPLY IF YOU DON'T UNDERSTAND. If you don't understand my reply write back and tell me. This is very important if I have told you that I intend to publish your letter. The chances are excellent that if you don't understand my reply then others will not. If you let me know that you still don't understand then it gives me another chance to express the solution in different terms.

I am not sitting at home just waiting for your letters. I also have a living to earn. The idea behind answering enquiries is so that I can find out what problems members have. I can then try to cover them in NILUG NEWS. As I have already said I don't guarantee to answer your letters but I'll do my best.

Robert Poate.

Dear Mr Poate,

I joined the user group on my visit to The Computer Fair. It looks as if you have written the newsletter for the computer expert! I say that because having read through it only some of it makes sense to me! I would welcome a series on machine code. I have had a go at learning machine code before but gave up as I found the writers often assume the reader has a sound knowledge of logic and memories etc.

I have had my computer apart and found the empty socket. My computer does not have EPROMs but has the usual ROMs in it. I would like to store 'PUNJABI' script in it. I have access to an eprom programmer and have sufficient knowledge to define the character set. I do not know how it can be accessed from the keyboard direct, so that each key press produces a character on the screen, perhaps you could help. Hoping to hear from you.

Yours sincerely

D.S.Jittla.

The key to the solution of your problem lies in knowing how the LYNX knows how to form the characters it displays. It has all the characters defined in two tables. The first holds the characters shown in appendix 3 in the manual and the second hold the characters shown on page 56.

Each character is defined by ten bytes and each character is in numeric order within the tables. There are two pointers to the tables. They are located at 626FH and 6271H. Now it is possible to change these pointers to point to your table of characters.

There are 256 possible characters and they are numbered 0-255. It is usual to split them into three blocks. The first is from 0-31. These are used as special characters and on the LYNX they are associated with the VDU command. Consequently characters do not appear on the screen when they are output. The output routine detects that they are special and performs the appropriate action. The second block is from 32-127 and these are the normal ASCII codes (Apendix 3). The third block runs from 128 to 255. These are normally graphics characters. On the LYNX only a few are present 224-247 (page 65).

The first pointer points to 0094H which is where the character 00 would be found. On the LYNX character 00 does not exist neither do the characters 1-31 consequently the actual table for the characters starts at 01D4H ie 320 bytes further on from 0094H. The graphics pointer points to the character 128.

So to set up extra characters in an eprom simply define the characters in 10 byte blocks starting with the first byte in the eprom. When the eprom is in position it will be at 4000H. To get the new characters to come up with the normal keys the first pointer needs to be changed. This can be done using the DPOKE command. Since it has to point to a location 320 bytes before the actual first character I think you will find that

```
DPOKE ALPHA,&4000-320
```

will do the trick. To set it up so that the extra characters may be accessed using the graphics mode (CONTROL 1 see page 55) the graphics pointer needs to be set to point to the new characters. Since it points at the first definded character

```
DPOKE GRAPHIC,&4000
```

should do the trick.

Dear NILUG,

Can you help? I have bought a composite video monitor (a TRANSTEC 1200) to use with my LYNX. However this has a phono input while the LYNX has a 5-pin DIN output. Can you tell me which pins from the LP socket need to be used to get a display on the monitor and where on the phono end should these be connected?

Many thanks,

Simon Rudd.

This only goes to show that you live and learn. I own a composite video monitor and I hadn't spotted that pin 4 of the light pen socket is a composite video signal. The answer is quite simple. Connect pin 4 to the center pin of the phono and connect pin 3 to the outside of the phono.

Review of GEMPACK 4 by GEM SOFTWARE Ltd. Price £7.90

By Danny Allan

SUB CHASE

The idea is to depth charge the submarines, with a maximum of three depth charges on the screen at one time. The subs, ships and charges all move very slowly, but the explosions are good even without sound. An increase in skill level doesn't increase speed, only the amount that the sub dodges up and down and the number of missiles it fires. The only challenge to the game is when the sub is very close to the surface as the ship can't get away from the missile fast enough. It takes some thought on the 3 & 4 skill levels. Four out of ten.

SEA HARRIER

This is a much better game. The airplane has to bomb the clouds, because if it hits nine clouds it is destroyed. But if the clouds hit the ship (that you eventually have to land on) the ship shrinks and an indestructible pink cloud is formed. One touch of this special cloud by the plane and it is destroyed. Seven out of 10.

It is a shame that neither game has good sound effects. Value for money - five out of ten.

Reveiw of LYNX COMPUTING by Ian Sinclair. Published by Granada Price £6.95 By M. Lawson

When I bought my LYNX in early March, it was my first computer and consequently I knew absolutely nothing about programming it. In the ensuing months I found myself getting more and more frustrated, as I found the latter stages of the instruction manual less and less informative. When Ian Sinclair's book 'LYNX COMPUTING' became available I ordered it without delay, or even seeing it.

I found it a friendly book, written in a manner that did not try to blind you with science. I feel that it is well worth while buying the book when the computer is purchased.

The opening chapter explained the 'SETTING UP' instructions even more basically than the manual. It gives a foolproof guide to making the machine work. On the problem of loading the author claims, as does the manual, there are several cassette recorders in the '£20-£30 price range' that will work with the LYNX. I have tried several but could not 'SAVE' satisfactorily on any of them. Most of the ones Camputers recommended were either not available or over £40.

I found that after studying by day and typing by night, I was making steady progress. Reading 'LYNX COMPUTING' did give me a lot of handy tips regarding ideas, structuring and programme format. I did remember one problem I had. Trying to get the program on page 83 fig 7.2 to work. Was it me? I never did sort it out.

I did get the impression that Mr Sinclair must think that the LYNX is the greatest thing since creamed cheese! The way he extols the virtues of the machine and the Camputers staff. At the time of purchasing the book I did not particularly share his enthusiasm, as the lack of information, the various shortcomings of the manual and the software problems, left me feeling that I should have bought another machine. However, with the aid of 'LYNX COMPUTING' I have finally been able to appreciate the machine's qualities and have resolved that it is extremely good value for money.
One point well worth bearing in mind is that a colleague of mine recalls the launch of the BBC models. He remembers the similar frustrations of people who bought those machines. Just look at the software and peripherals available for them today.

I digress, back to the book. For me where the book really scored was the comprehensive way that SOUND and GRAPHICS were explained. Well done sir, your explanation was clear enough for me to be a dab hand at graphics now. The many useful charts and graphs drawn in the book, enabled me take photocopies, so that I may now experiment with ease.

Thank you for also for the Appendix A regarding 'Cassette Loading Problems'. My neighbour is also glad, as he no longer has to loan me his cassette recorder whenever I buy a new pre-recorded tape.

To sum up, I would say that 'LYNX' and 'LYNX COMPUTING' go hand in hand. A fine book for a fine machine. Now, how about a similar book on machine code!

Reveiw of LYNX COMPUTING by Ian Sinclair. Published by Granada Price £6.95 By R.B.Poate

In spite of what it says on the back cover, this book is clearly aimed at the beginner and the absolute beginner at that. This doesn't mean to say that the expert will learn nothing from reading it.

Chapter 1 deals with setting up the LYNX. I found it hard to believe that anyone needs to be told how to put the LYNX together. There are only two leads and all the plugs are unique so how could you go wrong?

Chapters 2 to 5 deal with the bulk of the BASIC language as implemented on the LYNX. The order in which the elements are introduced is logical and they are supported with small programs for the reader to try.

I thought chapter six was excellent. It deals with the concept of subroutines and procedures. Having spent a lot of time helping beginners sort out their programs I now wish that they had read this chapter. The mistake that most of them make is to believe that computer programs MUST be written in a computer language. Ian Sinclair clearly demonstrates that you should start writting your programs in ENGLISH. You then proceed step at a time towards the final program written (in this case) in BASIC. Its just a pity the example chosen wasn't slightly better implemented.

Chapter 7 introduces some more elements of BASIC. I couldn't understand what a Shell-Metzner sort program was doing at this point. It simply appears and with little explanation. I think Ian lost his way a little. It would have been much better to have a simple inefficient bubble sort with a good explanation. After all this is a book for beginners.

The last three chapters deal with graphics and sound. This is where I started to learn. Although I've been around computers for over a decade (I started on paper tape!) I've never had a computer with high resolution graphics or sound.

On the whole a good book for the absolute beginer to buy. If the LYNX is your introduction to computing then this book would be money well spent. The fundamental mistake beginners may make is to read the book. I know books are written to be read but this book should be WORKED through, cover to cover. If after that you don't understand what LYNX Basic can do you never will. Its a pity that chapter 6 on program design didn't come at the end. I wonder if the beginner will be able to realize that this book is really about what the LYNX's BASIC can do and not how to do it. If the last chapter had been an introduction to using the BASIC then the beginner would at least know that he still had a long way to go.

If your're an expert then you may be able to learn something from this book but I doubt its worth buying.

Now if you're neither beginner nor expert then toss a coin or better still nip along to your nearest stockist and have a browse through it. LYNX COMPUTING by Ian Sinclair is definitely worth a read. Whether its worth buying only you can decide.

OTHELLO and SECTORS By Chris Cytera

OTHELLO is the board game otherwise known as REVERSI. The object of the game is to create as many pieces of your own as possible. This is done by surrounding your opponents pieces by placing a piece so that rows of your opponents pieces are bounded by one of your own at either end. Horizontal, vertical and diagonal rows may be surrounded. Moves which do not surround any oppsing pieces are illegal. The game ends when either the board is full, or no legal moves are possible. In this version of the game you are playing the computer. Enter your moves as the row number followed by the column letter, eg. 3D. If you cannot move, enter a '-' (minus sign). This can be used to make the LYNX take the first turn.

SECTORS draws a spiral which is made up from solid sectors (ie triangles). The sector angle is the angle in degrees subtended by each sector at the center of the spiral. The radius change is the rate at which the sectors spiral inwards. Both of these parameters are variable by the user. Values of radius change less than sixteen tend to give the best results.

```
10 PROTECT 0
20 VDU 1,GREEN,2,BLACK
,4,24
30 PRINT  TAB 17;
"SECTORS";CHR$(25);
40 PRINT  @ 3,45;
"Sector angle";
50 INPUT S
60 INPUT
"Radius change";D
120 LET R=500
130 LET A=0
140 LET S=RAD(S)
150 CLS
160 LET C=1
170 MOVE 128+R/5,124
180 REPEAT
190   INK C MOD 7+1
195   LET C=C+1
200   PROC PLOT
210 UNTIL R<=D
220 LET A$=GET$
230 REPEAT
240   OUT &0080,8
250   PAUSE 1000
260   OUT &0080,4
270   PAUSE 1000
280   OUT &0080,0
290   PAUSE 1000
300 UNTIL KEYN=32
310 RUN
320 DEFPROC PLOT
330 LET R=INT(R-D)
340 LET A=A+S
350 LET P=R*COS(A)/5+
128,Q=R*SIN(A)/4+124
360 PROC TRI1(128,124,
P,Q)
365 MOVE P,Q
370 ENDPROC
97999 REM TRI.FILL V3.2
98000 DEFPROC TRI1(a,b,
c,d)
98010 PROC TRI2(a,b,c,d,
PEEK(&6265),PEEK(&6267))
98020 ENDPROC
98030 DEFPROC
TRI2(a,b,c,d,e,f)
98040 PROC SORT
98050 IF d>f THEN LET
i=d,d=f,f=i,i=c,c=e,e=i
98060 PROC SORT
98065 LET b=INT(b),
d=INT(d),f=INT(f)
98070 IF d=b THEN
PROC HRZBAS
98080 ELSE  PROC SLPBAS
98090 ENDPROC
98100 DEFPROC SLPBAS
98110 LET q=(e-a)/(f-b),
p=(c-a)/(d-b)
98120 IF c>e THEN
SWAP p,q
98130 LET h=a,x=a
98140 PROC FILL(b,d,0)
98150 IF f=y THEN
ENDPROC
98160 LET h=h-p,x=x-q
98170 LET p=(e-h)/(f-y),
q=(e-x)/(f-y)
98180 PROC FILL(d,f,1)
98190 ENDPROC
98200 DEFPROC HRZBAS
98210 IF f=b THEN
PROC LINE
98220 ELSE  PROC TRI
98230 ENDPROC
98240 DEFPROC LINE
98250 MOVE a,b
98260 DRAW c,d
98270 DRAW e,f
98280 ENDPROC
98290 DEFPROC TRI
98300 LET p=(e-a)/(f-b),
q=(e-c)/(f-d)
98310 LET h=a,x=c
98320 PROC FILL(b,f,1)
98330 ENDPROC
98340 DEFPROC SORT
98350 IF b>d THEN LET
i=b,b=d,d=i,i=a,a=c,c=i
98360 ENDPROC
98370 DEFPROC
FILL(j,k,n)
98380 FOR y=j TO k-n
98390   MOVE h,y
98400   DRAW x,y
98410   LET h=h+p,x=x+q
98420 NEXT y
98430 ENDPROC
```

```
100 PROTECT BLACK
110 VDU 1,7,2,BLACK,4
120 INPUT "WHAT IS
YOUR NAME";N$
130 H=ASC(N$),C=76,M=1
140 IF H=C THEN LET H=89
150 DIM A(99),N(7)
160 FOR S=0 TO 7
170   READ N(S)
180 NEXT S
190 VDU 4,31,31
200 PRINT TAB 12;"A B C
 D E F G H"CHR$(31)
210 FOR j=1 TO 8
220   PROC CU(9,2+j)
230   PRINT j;
240   PROTECT MAGENTA
250   PRINT " * * * *
 * * * *";
260   PROTECT BLACK
270 NEXT j
280 FOR j=0 TO 99
290   LET A(j)=0
300 NEXT j
310 LET T=C,P=44
320 PROC PIECE
330 LET P=55
340 PROC PIECE
350 LET T=H,P=45
360 PROC PIECE
370 LET P=54
380 PROC PIECE
390 LABEL PLR
400 PROC SCORES
410 IF (B$="-" AND M=0)
OR Y+Z=64 THEN
GOTO LABEL END
420 INPUT "YOUR MOVE";B$
430 IF B$="-" THEN GOTO
LABEL COMP
440 LET a=ASC(B$),
b=ASC(RIGHT$(B$,1))
450 IF a<&31 OR a>&38
OR b<&0041 OR b>&0048
THEN GOTO 490
460 LET P=VAL(B$)*10+b-
ASC("A")+1,T=H,F=0
470 PROC MOVE
480 IF S>0 THEN GOTO 520
490 PROC UP
500 VDU 30
510 GOTO 420
520 LET F=1
530 PROC MOVE
540 LABEL COMP
550 PROC SCORES
560 IF(B$="-"AND M=0)OR
Y+Z=64 THEN
GOTO LABEL END
570 LET M=0,D=0,F=0,T=C
580 ?"LET ME THINK....";
590 FOR U=1 TO 8
600 FOR V=1 TO 8
610 LET P=U+10*V
620 PROC MOVE
630 IF S<>0 THEN
PROC EVAL
640 NEXT V
650 NEXT U
660 VDU 30
670 IF M>0 THEN PROC Z
680 IF M=0 THEN
PROC NOMOV
690 GOTO LABEL PLR
700 DEFPROC EVAL
710 IF U=1 OR U=8 THEN
LET S=S*2
720 IF V=1 OR V=8 THEN
LET S=S*2
730 IF U=2 OR U=7 OR V=2
OR V=7 THEN  S=S DIV 2
740 IF S=D THEN
LET D=D-RAND(2)
750 IF S>D THEN D=S,M=P
760 ENDPROC
770 LABEL END
780 IF Y>Z THEN PRINT
"Congratulations -
you win"
790 IF Y=Z THEN PRINT
"** A DRAW **"
800 IF Y<Z THEN PRINT
"I win !"
810 ? "Another game?";
820 REPEAT
830   LET R$=GET$
840 UNTIL R$="Y"
OR R$="N"
850 IF R$="Y" THEN RUN
860 END
870 DEFPROC PIECE
880 IF T=H THEN INK 5
890 ELSE INK RED
900 PROC CU(2*(P MOD 10
)+9,P DIV 10+2)
910 VDU 239
920 PAUSE 1000
930 LET A(P)=T
940 VDU 8,T,1,WHITE
950 ENDPROC
960 DEFPROC SCORES
970 LET Y=0,Z=0
980 FOR j=11 TO 88
990   IF A(j)=H THEN
LET Y=Y+1
1000   IF A(j)=C THEN
LET Z=Z+1
1010 NEXT j
1020 PROC CU(0,12)
1030 VDU 13,13,13
1040 PROC CU(0,12)
1050 PRINT N$": "Y;" "
1060 ? "LYNX: "Z;" "
1070 ENDPROC
1080 DEFPROC MOVE
1090 LET R=P,S=0
1100 IF A(P)<>0 THEN
ENDPROC
1110 IF F=1 THEN
PROC PIECE
1120 FOR j=0 TO 7
1130   LET Q=R+N(j),
P=R,K=0
1140   WHILE A(Q)<>0
AND A(Q)<>T
1150     LET Q=Q+N(j)
1160     LET K=K+1
1170   WEND
1180   IF A(Q)=0 THEN
GOTO LABEL LOOP
1190   LET S=S+K
1200   IF F=0 OR K<1
THEN GOTO LABEL LOOP
1210   FOR L=1 TO K
1220     LET P=P+N(j)
1230     PROC PIECE
1240   NEXT L
1250   LABEL LOOP
1260 NEXT j
1270 ENDPROC
1280 DEFPROC CU(x,y)
1290 ? @ x*3+3,y*10+5;
1300 ENDPROC
1310 DEFPROC Z
1320 LET F=1,P=M
1330 PROC MOVE
1340 ENDPROC
1350 DEFPROC NOMOV
1360 ? "I can't move";
1370 PAUSE 500
1380 ENDPROC
1390 DEFPROC UP
1400 VDU 28,28,28,5
1410 ENDPROC
1420 DATA 11,10,9,1,-1,
-9,-10,-11
```

MACHINE CODE FOR BEGINNERS
========================

Introduction

This is the first in a series of articles on machine code for beginners. They are aimed at LYNX users who wish to learn machine code and assembler programming. It has been assumed that readers will understand simple programming concepts such as variables, subroutines etc.

It must be pointed out that there is no such thing as an armchair programmer in the same way that you can be an armchair football supporter. If you wish to learn then you must try the example programs and have a go at any suggested modifications.

You can't go very far in computers before you bump into numbers so lets start with them.

Number Systems

Before we look at any new numbers systems lets consider a system that you are already familiar with - DECIMAL. This system uses the ten digits 0-9 with which to represent numbers. As I'm sure you know counting goes like this :-

0,1,2,3,4,5,6,7,8,9,10,11,12,13 ... etc.

When you count the process that is being performed is as follows :-

You start at zero	0
Use the next digit	1
Use the next digit	2
.	.
.	.
Continue until the last digit.	9
When you run out of digits reset the column and carry one across	10
Use the next digit	11
etc	

Another way to look at decimal is to consider what each figure in each column represents. The first three columns are hundreds tens and units. So any number is made up by deciding how many of each column you need.

	100	10	1		
1 is	0	0	1	ie 0x100 + 0x10 + 1 =	1
2 is	0	0	2	ie 0x100 + 0x10 + 2 =	2
20 is	0	2	0	ie 0x100 + 2x10 + 0 =	20
21 is	0	2	1	ie 0x100 + 2x10 + 1 =	21
134 is	1	3	4	ie 1x100 + 3x10 + 4 =	134

I know you knew all of this but the point is that all the number systems that are needed for this series work in exactly the same way. Once you have mastered one system (which you have) there should be no problem with the others.

So now lets have a look at BINARY numbers. If to count in tens (decimal) you need ten digits then to count in twos you need two digits and the digits 0 and 1 are used. To count is exactly as above.

	Binary		Decimal
We start at zero	0	=	0
We use the next digit	1	=	1
When we run out of digits we carry one across and reset the previous column ...	10	=	2
Now we increment again	11	=	3
When we run out of digits we carry one across. This time we carry one across from two columns	100	=	4
We use the next digit	101	=	5
We increment which forces a carry	110	=	6
We increment	111	=	7
When you run out of digits you carry one across. This time we need to carry one across in three columns	1000	=	8
	etc		

Again we can look at binary by considering what each column represents. Instead of hundreds, tens and units we have 8s, 4s, 2s and 1s. Hence counting looks like this:-

Binary	8	4	2	1			Decimal
0000 is	0	0	0	0	ie 0x8 + 0x4 + 0x2 + 0x1	=	0
0001 is	0	0	0	1	ie 0x8 + 0x4 + 0x2 + 1+1	=	1
0010 is	0	0	1	0	ie 0x8 + 0x4 + 1x2 + 0x1	=	2
0011 is	0	0	1	1	ie 0x8 + 0x4 + 1x2 + 1x1	=	3
0100 is	0	1	0	0	ie 0x8 + 1x4 + 0x2 + 0x1	=	4
0101 is	0	1	0	1	ie 0x8 + 1x4 + 0x2 + 1x1	=	5
0110 is	0	1	1	0	ie 0x8 + 1x4 + 1x2 + 0x1	=	6
0111 is	0	1	1	1	ie 0x8 + 1x4 + 1x2 + 1x1	=	7
1000 is	1	0	0	0	ie 1x8 + 0x4 + 0x2 + 0x1	=	8
1001 is	1	0	0	1	ie 1x8 + 0x4 + 0x2 + 1x1	=	9

etc.

O.K. so much for binary now HEXADECIMAL. HEX means six and DECIMAL means ten so hexadecimal means 16. It should come as no surprise to you to find that we need 16 digits to be able to form the numbers. Now we have ten of them because we can use 0-9 what we now need is six more. It seems quite logical to use A,B,C,D,E and F. Counting is exactly as before. You start at zero and keep using the digits until you exhaust them. After that you carry one across into the next column, reset the original column and carry on. So numbers in hexadicimal look like this.

	Hex		Decimal
Start at zero	0	=	0
Use the next digit	1	=	1
Use the next digit	2	=	2
Continue	.		
.	.		
.	9	=	9
.	A	=	10
.	B	=	11
.	C	=	12
.	D	=	13
.	E	=	14
.	F	=	15
When you run out reset the column and carry one across ...	10	=	16
Use the next digit	11	=	17
Use the next digit	12	=	18
Continue	13	=	19
	etc.		

Again you can consider what each column represents. With hexadecimal numbers the columns go 4096, 256, 16 and 1.

```
Hex      4096| 256| 16| 1                                  Decimal
0004 is    0|    0|  0| 4  ie 0x4096+0x256+0x16+4x1   =        4
000C is    0|    0|  0| C  ie 0x4096+4x256+0x16+12x1  =       12
002A is    0|    0|  2| A  ie 0x4096+0x256+2x16+10x1  =       42
01A2 is    0|    1|  A| 2  ie 0x4096+1x256+10x16+2x1  =      418
043C is    0|    4|  3| C  ie 0x4096+4x256+3x16+12x1  =     1084
4E2A is    4|    E|  2| A  ie 4x4096+14x256+2x16+10x1=     20010
```

Hexadecimal is a bit of a mouthful and not surprising -ly it is normally shortened to HEX. Numbers are often given the suffix H when written to distinguish them from decimal numbers. So 10 is ten but 10H is sixteen. Binary numbers are nearly always written as 8-bit numbers like:- 01010110, consequently there is seldom any confusion.

A few more concepts need to be explained before we start programming. Micro-computers (like the LYNX) work on logic. Logic is either TRUE or FALSE which in machine code terms becomes a 1 or a 0. Computers store logic in memory. Storage is measured in BITS which is a word formed from BInary digiTS. Bits are the smallest unit of logic storage. They may either take the value 1 or 0. The unit becomes very cumbersome and so bits are grouped into BYTES. A byte consists of 8 bits. Bytes is also a small unit and so the concept of KILOBYTE is used. As you are probably aware kilo means 1000. In computer terms the kilo actually means 1024. This is because 2 to the power of 10 is 1024, ie its the nearest binary power to 1000. Kilobytes is normally abreviat -ed to K. So when one talks about a 48K LYNX we know that it has:-

48 x 1024 = 49152 bytes of memory.
49152 x 8 = 393216 bits of memory.

There are basically two types of memory. These are RAM and ROM. Ram is Random Access Memory. It allows the programmer to read and to write to any part in it. ROM, on the other hand, is Read Only Memory. You may only read what is there and you may not change it. There are various variants of ROM, for example PROM and EPROM. For the purposes of understanding machine code you can consider them as ROM ie you can only read them.

In normal operation the Z80 will support 64K of memory. So that each byte of memory may be distinguished from any other byte all the memory locations have a unique ADDRESS by which they may be referenced. The first address is 0 and the last address is FFFFH or 65535 in decimal.

So much for memory now for the CPU. Inside the Z80 there are 208 bits of RAM. It is grouped into 8-bit and 16-bit registers. Registers differ from normal RAM in that special operations may be performed with them. The following diagram shows how the registers are grouped.

```
                                           +----+----+
                                           | I  | R  |
  +----+----+        +----+----+           +----+----+
  | A  | F  |        | A' | F' |           |   PC    |
  +----+----+        +----+----+           +----+----+
  | B  | C  |        | B' | C' |           |   SP    |
  +----+----+        +----+----+           +----+----+
  | D  | E  |        | D' | E' |           |   IX    |
  +----+----+        +----+----+           +----+----+
  | H  | L  |        | H' | L' |           |   IY    |
  +----+----+        +----+----+           +----+----+
MAIN REGISTERS     ALTERNATE SET        SPECIAL PURPOSE
                                           REGISTERS
```

There are 8 registers which make up the MAIN SET. The A register is known as the ACCUMULATOR and as its name suggests it generally accumulates the results of operations. The F register contains FLAGS which indicate the effect of the previous operations. The remaining registers B,C,D,E,H and L are general purpose registers. The main set may be paired into REGISTER PAIRS. The pairing is as shown in the diagram ie AF, BC, DE and HL.

The ALTERNATE SET of registers is exactly what its name implies an alternative set. Since anything that applies to the MAIN SET also applies to the ALTERNATE SET we need only consider the main set. Again the SPECIAL PURPOSE REGISTERS are exactly what their name implies, registers for special purposes. We will deal with all of them in time but for now we will just consider the PC or Program Counter. The PC is a 16-bit register which the Z80 uses to point to the next instruction to be executed.

Programs are sets of instructions which (generally) perform some useful action. To run a program the PC is loaded with the first byte of the program. The Z80 will then fetch the first instruction and process it. The PC is automatically altered (by the Z80) to point to the next instruction. This is then fetched and processed. This procedure of fetch and process goes on all the time. The instructions are fetched into the Z80 as 8-bit binary numbers or codes. Since they perform an operation they are know as OPCODES. Theoretically there are only 256 possible opcodes but in fact by using prefixes there are about 600.

Now if, as has been stated, the Z80 communicates in binary then it should be possible to program it in binary. In fact it is possible, consider the following program:-

```
00100001
01011011
01100010
01111110
00100011
01000110
01110111
00101011
01110000
11001001
```

If it were fed to the Z80 it would process it quite happily. How do you feel about writing programs in binary? Do you think you would spot an obvious mistake? Do you think you could handle all those 0s and 1s? Most people would find it impossible to program in binary. So the next step is to convert it into something a little easier for humans to handle - HEX. The conversion is quite simple and in fact after a little practice you will do it in your head.

Consider the byte 01011101. Now the trick is to split it into two parts thus 0101 1101. Each part is called a NIBBLE (a nibble is part of a byte!) Converting each nibble is quite straightforward.
Taking the left-hand nibble we have
0101 is 0x8 + 1x4 + 0x2 + 1x1 = 5.
Taking the right-hand nibble we have
1101 is 1x8 + 1x4 + 0x2 + 1x1 = 13.
In hex 5 is 5 but 13 is D. Hence 01011101 is 5DH. The largest number you will need to be able to handle is 1111 ie fifteen and most people can handle it in their head. Now back to our program which after conversion looks like:-

```
21
5B
62
7E
23
46
77
2B
70
C9
```

Now it may still not mean much to you but I think you will agree that it is a lot easier to handle than the binary. Hex then, is merely a way of making things easier for us humans.

Now if we want to use hex and the Z80 insists on using binary (and it does) then some form of translation is going to be needed. It is normally done by a HEX LOADER. Fortunately the LYNX has a hex loader. It is the Monitor M command. The M command allows us to examine any location in memory. If the memory is RAM then it will also allow us to change it. So we may now enter our program. The only problem is where? I have found that beginners find this one of the more difficult aspects of learning machine code and I intend to deal with it a little later. Just for now we will enter the program at 9000H.

So then to load it type MON [RETURN] to bring up the Monitor and then M9000 [RETURN] to start loading. Ignore the previous contents which are displayed between the greater than and less than symbols and type in the first opcode ie 21. After you have entered each opcode press RETURN and continue with the next. After the last one has been entered use a fullstop to end the M command. We are now ready to run our program. To do this we need to set the PC (program counter) to point to the first byte of our program. Fortunately this is easy to do because the Monitor has a command to do it for us. It is the G command. If you type G9000 [RETURN] you shoud see the results immediately. Try running the program a few times.

O.K. so whats going on? The Z80 takes the opcodes in turn starting with the first ie 21. This opcode tells the Z80 to take the two following bytes (ie 5B & 62) and load them into the HL register pair. Now unfortunately, this is not quite as straightforward as it could be because the bytes are loaded in reverse order hence H ends up containing 62 and L ends up with 5B. HL now has 625B which you may know is the address of where the INK colour is stored. The 7E opcode loads what HL points to into the A register. Hence A now contains the INK colour. The 23 opcode increments HL. HL now contains 625C which is where the PAPER colour is stored. The 46 opcode tells the Z80 to load the B register with whatever the HL register pair points to. In other words the PAPER colour is loaded into the B register. The 77 opcode loads the A register to wherever HL points ie the INK colour is placed where the PAPER colour is stored. The HL register pair is decremented by the opcode 2B so HL now points to the store for the INK again. The B register is loaded to where the HL register pair points to by the opcode 70, hence the PAPER colour has been put into the INK colour store. The final C9 opcode is a return instruction. As we had come from the Monitor that's where we return to . These 10 byte are an inverse video routine or 'VDU 18'.

So we can summarise the program as follows:-

```
21 5B 62  Load HL with 625BH.
7E        Load A with what HL points to.
23        Increment HL.
46        Load B with what HL points to.
77        Load A to where HL points.
2B        Decrement HL.
70        Load B to where HL points.
C9        Return to where you came from.
```

Before I finish with this program the final point is to demonstrate how you can include it in a Basic program. The LYNX supports machine code routines with the reserved words CODE, LCTN and CALL. CODE allows you to enter opcodes in your Basic program. They may then be executed by CALL. So if you enter:-

```
10 CODE 21 5B 62 7E 23 46 77 2B 70 C9
20 CALL LCTN(10)
```

and then run the program you will have and inverse video routine. You could always use VDU 18 but its not half as much fun!

Now I want to discuss a couple of questions that I know beginners find dificult to understand. The first is how did I know which registers to use? The answer is experience. It is an unhelpful answer because beginners don't have any experience so they are in a catch 22 situation. The way out of it is to try the examples and UNDERSTAND what is going on. Once you have grasped the concept that the Z80 takes opcodes from memory and that the opcodes tell the Z80 what operation to perform you have won half the battle. After you have played with a few programs written by someone else you will soon get the hang of it. There are books available giving all the opcodes but I suggest you wait a little before you buy one.

The second question is how did I know where to load the program. The answer to this is a little more helpful. Quite simply you have to understand the LYNX's MEMORY MAP. A memory map shows what is where. Once you know where Basic is, what workspace it requires, where it stores variables and the user's program you will then see where the spare RAM is. So here is a memory map for the 48K LYNX.

```
     +--------------+
FFFF ! EXPANSION    !
     !              !
     !              !
A000 !              !
     +--------------+
9FFF ! USER RAM     !
     !              !
     !              !
6000 !              !
     +--------------+
5FFF ! EMPTY EPROM  !
4000 ! SOCKET       !
     +--------------+
3FFF ! LYNX BASIC   !
     !              !
     !              !
0000 !              !
     +--------------+
```

Now as you can see it is basically in four parts. First (starting at the bottom of the diagram) comes the Basic. It runs from 0000 to 3FFF. The 4000 to 5FFF slot is empty. User RAM starts at 6000 and runs up to 9FFF. Finally there is room for expansion from A000 up to FFFF. Since you can only put your programs in RAM we need to look at the USER RAM section in a little more detail. The following map gives the details.

```
     +----------------------------------+
9FFF !                                  !
     ! HIMEM                            !
     +----------------------------------+
     ! Stack working downwards          !
     !                                  !
9000 !                                  !
     !                                  !
     !                                  !
     !                                  !
     ! String and vector variables      !
     ! start at the end of the          !
     ! program     and work upwards     !
     ! - - - - - - - - - - - - - - - -  !
     ! Basic program ends somewhere     !
     !                                  !
     !                                  !
     !                                  !
     ! Basic program starts             !
694D ! here and works upwards           !
     +----------------------------------+
     ! Pointers to string variables     !
68E4 ! A$,B$,C$ etc                     !
     +----------------------------------+
     ! Pointers to vector variables     !
6814 ! A(20),C(34) etc                  !
     +----------------------------------+
     ! Basic scalar variables           !
670C ! A,B,C etc                        !
     +----------------------------------+
     ! Basic workspace                  !
     !                                  !
6000 !                                  !
     +----------------------------------+
```

Now this map shows that there is something called a stack which works its way down from 9FFF and that a Basic program works its way up from 694D. The rest of the RAM space is taken up by Basic as workspace of one sort or another. If you are only using machine code then where in this block of RAM you load the program doesn't really matter. Hence 9000 was as good a place as anywhere else. If however you want to use machine code in conjunction with Basic then you have to ensure that your programs don't clash. This is best done by using RESERVE to move the HIMEM marker down. HIMEM tells Basic how much memory it may use. By bringing it down you can set aside memory for machine code programs. For example,

```
        RESERVE HIMEM-300 [RETURN]
```

would set aside 300 bytes for a machine code program. The USER RAM memory map would then look like this:-

```
     +----------------------------------+
9FFF !                                  !
     ! 300 bytes which Basic will not   !
     ! use. It may be used for machine  !
     ! code routines.                   !
     !                                  !
9ECC ! HIMEM                            !
     +----------------------------------+
     ! Stack working downwards          !
     !                                  !
     !                                  !
9000 !                                  !
     !                                  !
     !                                  !
     !                                  !
     !                                  !
     ! String and vector variables      !
     ! start at the end of the          !
     ! program     and work upwards     !
     ! - - - - - - - - - - - - - - - -  !
     ! Basic program ends somewhere     !
     !                                  !
     !                                  !
     !                                  !
     !                                  !
     ! Basic program starts             !
694D ! here and works upwards           !
     +----------------------------------+
     !                                  !
```

Programs could now be loaded at 9ECC and Basic would not corrupt them.

The next program is going to be a routine to put 'DISPLAY YOUR NAME' onto the screen. Now this is where you start to learn not to re-invent the wheel. There is already a routine in Basic that will put characters onto the screen for you. Details of how characters are put onto the screen were described in Issue 1 of NILUG NEWS so I don't intend repeating them here. All you need to know is that if you load the A register with the ASCII code for the character you want displayed and then call a routine at 06A4 the character will be displayed at the current cursor location.

This address may be found at location 6200H and may be displayed by typing PRINT hash DPEEK(&6200) [RETURN]. (Note I don't have a hash symbol on my printer). So the logic of the program looks like this:-

1. Load HL with the start of the bytes to be output minus 1
2. Increment HL to point to the next byte.
3. Load what HL points to into the A register
4. Call the output routine
5. Test for the end and go back to step 2 if not the end
6. Return.

Now we know how to do step 1. To load HL with an address you use the opcode 21 and follow it with the address bytes. Step 2 is also easy you use opcode 23. Likewise step 3 has already been done before you use opcode 7E. Now step 4 hasn't been done before. To call a routine you use CD llhh. Where CD is the opcode for CALL and llhh is the two byte address of where the call is to be made with the low byte (ll) before the high byte (hh). So to call 06A4 you need CD A4 06. Leaving out step 5 for the moment step 6 has already been done. You use the opcode C9.

So that only leaves us step 5. For this you need to know a new opcode which is 10 nn. This opcode performs two operatioons. Firstly it decrements the B register. Then it tests to see if the B register is zero. If it is then the Z80 will process the next instruction. If it isn't zero then a jump of nn bytes is made. So if we load B with the number of bytes to be displayed we could use this opcode.

The program now looks like this:-

```
Step 1  21 ll hh   Load HL with address of 'DISPLAY YOUR
                   NAME' minus 1.
        06 nn      Load B with the number of bytes.
Step 2  23         Increment HL
Step 3  7E         Load A with what HL points to.
Step 4  CD 88 62   Call output a character routine.
Step 5  10 nn      Decrement B and if not
                   zero jump back to step 2.
Step 6  C9         Return.
```

Now the next problem is to fill in the missing numbers. Firstly we need to decide where we are going to put the characters 'DISPLAY YOUR NAME'. I suggest on the end of the program. The next thing is to decide where the program is going to reside. I suggest 9000 again. This means that the address of the first byte of 'DISPLAY YOUR NAME' will be 9000+ number of bytes of program ie 9000+D (D is 13 remember?) So the address is 900D. But we wanted one less than this address so that when we increment in the loop we are pointing to the first character. Hence the address required is 900C. Therefore the first line of our program now becomes 21 0C 90 (note that the address is back to front). Now for the number of characters. I make 'DISPLAY YOUR NAME' to be seventeen characters. The quotes don't count and there are two spaces. Now seventeen in hex is 11H so line 2 now looks like 06 11. This only leaves the jump associated with the new opcode 10. This jump is a relative jump ie it is made relative to where we are. Now all jumps are made by changing the Program Counter. You will remember that it is the PC which tell the Z80 where to pick up the next instruction from. So before you know how many bytes the jump has to be you have to know what the PC will be pointing to when this instruction is processecd. When the Z80 returns from the output routine the PC points to the 10 opcode. This is then fetched and then the PC points to the nn byte. The nn byte is then fetched and the PC points to the C9 opcode. So when the jump is made the PC points to the C9 opcode. Now since the nn byte can only take values from 00 - FF and both positive and negative jumps are required the Z80 designers decided that positive jumps would be represented as 01, 02, 03 etc and negative jumps would be FF for -1, FE for -2, FD for -3 etc. Since the PC points to the C9 byte then FF would jump us back one byte to the nn byte, FE to the 10 opcode, FD to the 06 byte, FC to the A4 byte, FB to the CD opcode, FA to the 7E opcode and F9 to the 23 opcode.

So F9 would give us the required jump. The program is listed below. Note that I have put the hex addresses of where the opcodes should be on the left-hand side of the listing.

```
9000    21 0C 90   Load HL with address of 'DISPLAY YOUR
                   NAME' minus 1. (900C).
9003    06 11      Load B with the number of bytes.
9005    23         Increment HL
9006    7E         Load A with what HL points to.
9007    CD A4 06   Call output a character routine.
900A    10 F9      Decrement B and if not
                   zero jump back to step 2.
900C    C9         Return.
900D    44         ASCII for D in hex.
900E    49         ASCII for I
900F    53         ASCII for S
9010    50         ASCII for P
9011    4C         ASCII for L
9012    41         ASCII for A
9013    59         ASCII for Y
9014    20         ASCII for space
9015    59         ASCII for Y
9016    4F         ASCII for O
9017    55         ASCII for U
9018    52         ASCII for R
9019    20         ASCII for space
901A    4E         ASCII for N
901B    41         ASCII for A
901C    4D         ASCII for M
901D    45         ASCII for E
```

Now the idea behind this program was to get you to play around with it. First of all try changing 'DISPLAY YOUR NAME' to whatever your name is. Remember that the ASCII codes are in hex and not decimal as shown in appendix 3. Secondly try putting a leading 04 byte in front of the text. This will clear the screen. Next try putting an 07 byte in the string. This will sound the BEEP. You can try other VDU control codes from page 62. Finally try to rewrite the program to work backwards. Point HL to the end of the string of characters and DECREMENT HL instead of incrementing it. The opcode to decrement HL is 2B. This should put your name up backwards.

Have fun!

PARALLEL PORTS FOR THE LYNX

By Kenneth M'Alpine

This article describes a simple circuit which, when connected to the LYNX, will provide 3 input/output ports. Obviously, the applications of such a circuit are many, one being a printer driver as mentioned by the Editor in Issue 1. The circuit shown uses an 8255 PPI (Parallel Peripheral Interface), two standard logic IC's and a few passive components which depending upon the enclosure used, should cost you less than £10.

Circuit Description

Internally the 8255 has 4 registers which are: port A, port B, port C and a control register. These are selected by A1 and A2 while A0 determines $\overline{RD}$ or $\overline{WR}$ and thus the data transfer direction. The $\overline{RD}$ and $\overline{WR}$ signals from the LYNX are not active during IN or OUT instructions.
The IC is selected using the $\overline{CS}$ pin when $\overline{IORQ}$ (In/Out ReQuest), A4, A5 are low and A3, A6, $\overline{INT}$ (maskable Interrupt) are high. The INT line is included to ensure that the ports are disabled during the hardware interrupt which is generated by the cursor pin of the 6845, thus preventing problems on the data bus.
Notice that the address lines can be connected directly to the multiplexed address outputs from the LYNX as only A0-A6 and A13 are available during IN's and OUT's. A13, however, is not decoded so a reflection of the ports occur when this is high. On switch-on a short positive going pulse is applied to the reset pin which configures all three ports as inputs. C1 decouples the supply.

The decoding just described provides the following port addresses:-

DECIMAL	HEX	PORT	DECIMAL	HEX	PORT
72	48H	Write to port A.	73	49H	Read from port A.
74	4AH	Write to port B.	75	4BH	Read from port B
76	4CH	Write to port C.	77	4DH	Read from port C.
78	4EH	Write to control.	79	4FH	Not used.

Circuitry

The prototype was built on a small piece of Veroboard approx. 3.5x2.5 inches and enclosed in a small plastic box to protect the connections. An IDC 40 way socket was used to connect the 'interface' plug on the LYNX and joined, via 40 way ribbon cable, to the PPI circuit. Outputs from the ports were via 10 way molex latch plugs, providing +5v, eight data lines and ground for each port. The output connections could, however, be made using a 25 way D-type which could provide the eight data lines for each port plus a reference ground.

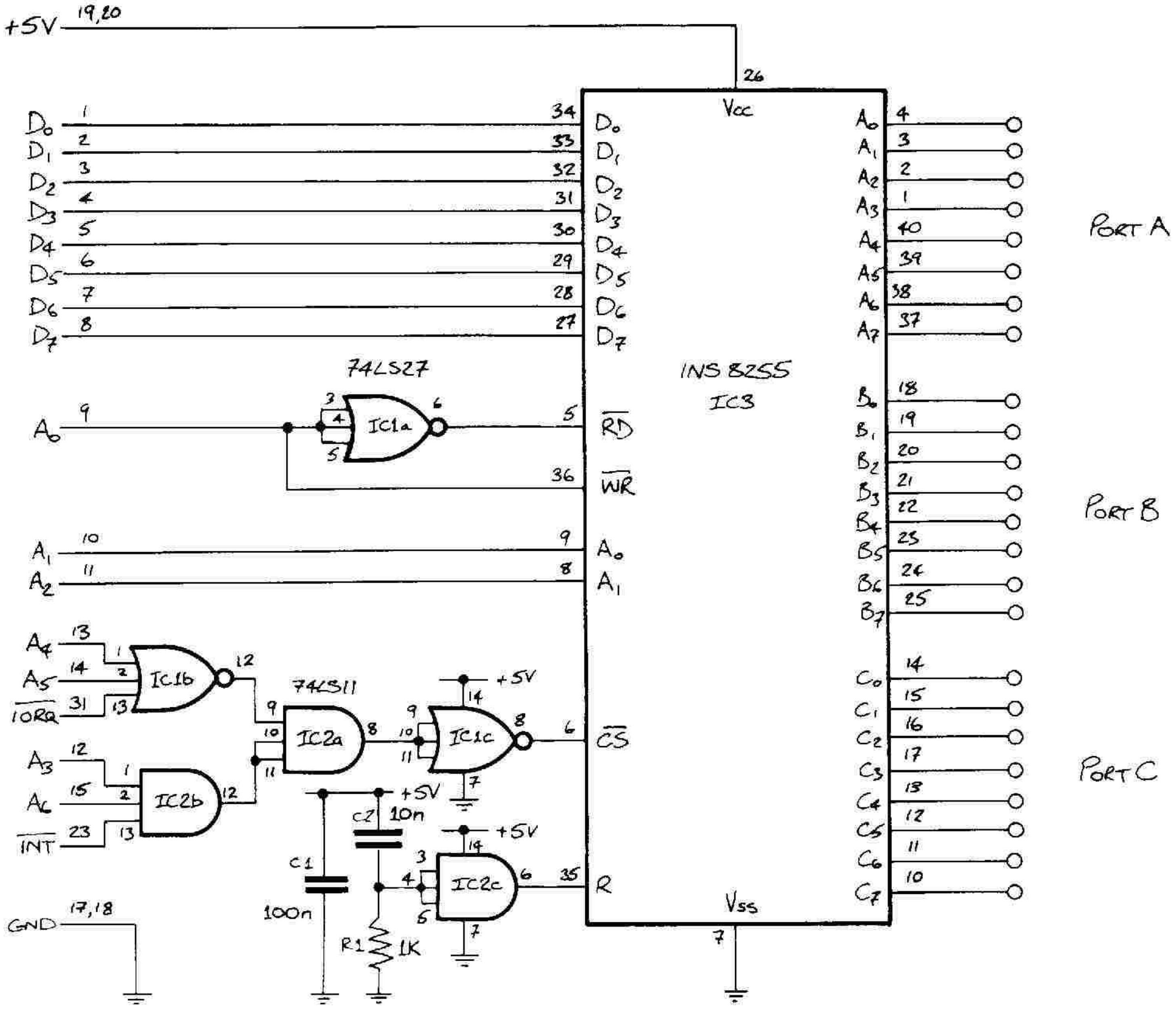
Using the Interface

Three modes of operation are possible with the 8255: mode 0, 1 and 2 (see 8255 datasheet). The most common, mode 0, allows port A and B to be input or output and port C to be split into two sections of 4 bits, each of which can be input or output. Configuration, and the control data required, are shown in the table opposite. As an example let us set up Port A to output, Port B to input and port C to output. Control data required is 130 (82H).

```
Enter  OUT 78,130 or via the monitor Enter O 4E 82
To send data to port A (or port C)
Enter OUT 72,xx (or OUT 76,xx) or via the monitor O 48 xxH (or O 4C xxH)
To read data from port B
Enter PRINT INP(75) or via the monitor Q 4B.
```

COMPONENTS	Resistors	ICs	Capacitors	Miscellaneous
	R1 1K	IC1 74LS27	C1 100n Polycarbonate	Veroboard
		IC2 74LS11	C2 10nF Polycarbonate	Enclosure to suit.
		IC3 INS 8255		IDC 40-Socket & cable
				Sockets for ports.

You can now unleash the power of your LYNX on the outside world with suitable peripherals connected.
Thanks go to David Barnes of G.W.D.S. for information about the LYNX interface.



CONTROL WORD		PORTS			
DECIMAL	HEX	A	B	C (UPPER)	C (LOWER)
128	8ϕ	OUT	OUT	OUT	OUT
129	81	OUT	OUT	OUT	IN
130	82	OUT	IN	OUT	OUT
131	83	OUT	IN	OUT	IN
136	88	OUT	OUT	IN	OUT
137	89	OUT	OUT	IN	IN
138	8A	OUT	IN	IN	OUT
139	8B	OUT	IN	IN	IN
144	9ϕ	IN	OUT	OUT	OUT
145	91	IN	OUT	OUT	IN
146	92	IN	IN	OUT	OUT
147	93	IN	IN	OUT	IN
152	98	IN	OUT	IN	OUT
153	99	IN	OUT	IN	IN
154	9A	IN	IN	IN	OUT
155	9B	IN	IN	IN	IN

DEFINITION OF MODE ϕ PORT CONTROL.

SCROLLING THE SCREEN ON A 48K LYNX

As you will know doubt have found out, the screen on the LYNX doesn't scroll. I'll grant you that you can play around with registers 12 & 13 of the 6845 but it's not proper scrolling. What I wanted to see was the screen scroll in the PROPER manner.

Screens are normally scrolled by moving the contents of the video ram up by one line and then blanking the bottom line. This action can be done on the LYNX provided that the program is in eprom. The LYNX's screen is composed of three banks of ram, one for each of the primary colours. In order to scroll the screen it would be necessary to move all three banks. Although this is possible it is so slow that it would be unacceptable in use. When I'm 'working' (should I say playing?) I normally use the TEXT mode (ie GREEN ink, black paper and protect red and blue). It seemed, therefore, quite reasonable to just scroll the green bank.

The first problem is to detect when a line feed occurs. This is easy since the line feed routine makes a call to 628BH. This is normally set to a RET (ie C9) instruction. It could be set to point to the scroll routine. The scroll routine must decide if a scroll is necessary. It does this by testing the Y component of the cursor's position. If it is greater than 234 then a scroll is required. The scroll routine works by moving ALL the green bank. It is not window sensitive so I decided that I should check that the full window area is being used.

I decided to locate the initialisation routine at 402DH because this means that the DISK command may be used to initialise it. At first I simply had the initialisation routine set up a jump to the SCROLL routine. This worked O.K. but there was a side effect. When you edit long program lines on the bottom line of the screen the edit routine goes wrong (its not the edit routine's fault). The problem is that it hasn't been told that the line has been scrolled. Consequently I decided to make the DISK command into an ON/OFF switch. The first time you type DISK the scroll routine is set on. The next time you type DISK it is set off. When I want to list I set the scroll on. When I want to edit I set it off.

```
4000          0010        ORG  £4000 ;Spare eprom socket
4000 0006     0020 VER    EQU  6
              0030 ;SCROLL FOR THE 48K LYNX
              0040 ;By R.B.Poate 14/7/83
              0050 ;
4000 402D     0060 DISK   EQU  £402D ;DISK execution add
4000 628B     0070 LFCALL EQU  £628B ;Line feed call add
4000 06       0090        DEFB VER ;Set version number
402D          0110        ORG  DISK
              0120 ;Flip scroll on or off
402D 214140   0130        LD   HL.SCROLL
4030 228C62   0140        LD   (LFCALL+1).HL
4033 218B62   0160        LD   HL,LFCALL
4036 3EC3     0170        LD   A,£C3
4038 BE       0180        CP   (HL)
4039 2802     0190        JR   Z,OFF
403B 77       0200        LD   (HL),A ;Set scroll on
403C C9       0210        RET
403D 3EC9     0220 OFF    LD   A,£C9
403F 77       0230        LD   (HL),A ;Set scroll off
4040 C9       0240        RET
When you call this scroll routine the registers are set :-
DE Y and X coordinates of cursor location
HL Y direction of window size
A  Y coordinate of the cursor location
              0340 ;Test if full window size is in use
4041 0105F5   0350 SCROLL LD   BC,£F505 ;Full window size
4044 AF       0360        XOR  A
4045 E5       0370        PUSH HL ;Protect HL
4046 ED42     0380        SBC  HL,BC
4048 E1       0390        POP  HL
4049 7A       0400        LD   A,D ;Restore the value of A
404A C0       0410        RET  NZ ;Return if not full
Scroll necessary? ie are we at the bottom of the screen?
404B 3EEA     0450        LD   A,234
404D BA       0460        CP   D
404E 7A       0470        LD   A,D
404F 3801     0480        JR   C,NEEDED
4051 C9       0490        RET  ;Not necessary
```

```
              0510 ;A scroll is necessary
4052 D5       0520 NEEDED PUSH DE ;Protect DE
4053 3E65     0540        LD   A,£65
4055 017FFF   0550        LD   BC,£FF7F
4058 ED79     0560        OUT  (C),A
405A 3E40     0570        LD   A,£40
405C D380     0580        OUT  (£80),A
405E 3EE4     0590        LD   A,£E4
4060 D380     0600        OUT  (£80),A
              0610 ;Now the GREEN ram may be scrolled
4062 2100C0   0620        LD   HL,£C000 ;Green  offset
4065 54       0630        LD   D,H
4066 5D       0640        LD   E,L
4067 014001   0650        LD   BC,320
406A 09       0660        ADD  HL,BC
406B 01C01D   0670        LD   BC,7936-320
406E EDB0     0680        LDIR ;Scroll the screen
              0690 ;Now clear the bottom line
4070 AF       0700        XOR  A
4071 77       0710        LD   (HL),A
4072 54       0720        LD   D,H
4073 5D       0730        LD   E,L
4074 1B       0740        DEC  DE
4075 014001   0750        LD   BC,320
407B EDB8     0760        LDDR ;Clear the bottom line
407A 017FFF   0770        LD   BC,£FF7F
407D D380     0780        OUT  (£80),A ;Return user ram
407F ED79     0790        OUT  (C),A
              0810 ;Prevent the LYNX's eproms
              0820 ;from resetting the cursor location.
4081 26FF     0830        LD   H,£FF
4083 D1       0840        POP  DE ;Return D
              0850 ;Now subtract 10 from D because
              0860 ;10 is about to be added to it
4084 7A       0870        LD   A,D
4085 D60A     0880        SUB  10
4087 57       0890        LD   D,A
4088 C9       0900        RET
```

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 II 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**

LYNX 48K SOFTWARE

DISASSEMBLER (U11)
This basic utility program enables you to examine machine code programs in ROM or RAM in standard Z-80 mnemonics.
Features:— Full disassembler of relative and absolute jumps, option of disassembly in decimal or hexedecimal notation.

LABYRINTH (G11)
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.
Features:— Fast moves, help option, sound effects, great graphics, replay of route.

CHANCELLOR (G21)
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.
Features:— Full graphical output of progress (or otherwise). Chancellor rating at end of term based on your overall achievement. A thinking person's game.

OTHELLO (G31)
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.
Features:— Three levels of play, full rules and instructions given, demonstration game, option to set up the pieces to any position, replay of last game.

SPACE TREK (G41)
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.
Features:— Three levels of play, long and short range scans, phasers, photon torpedoes, constant display of your status report, colour graphics and sound effects, including signature tune!

Price £4.75 each inc. p&p cheque/P.O. to:

**QUAZAR COMPUTING DEPT. N.
17 TEG CLOSE, PORTSLADE, EAST SUSSEX.**

Other programs available — send for details.

```
*** SCROLLER ***
=================
Scrolls the screen properly.
Supplied in 2764 EPROM
with instructions.
£10.00
(inclusive)
10% Discount for NILUG members

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

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.

FREE OFFERS

FREE catalogue containing something of interest for all micro users —

if you send us a stamped addressed envelope . . . Software, hardware, dust covers, books, T-shirts, cassette leads, LOADing aids, test cassettes & tape head de-magnetisers etc.

FREE 4 page leaflet that explains tape head azimuth alignment . . .

if you order any of our products (tape head misalignment can cause LOADing problems). 48K LYNX SOFTWARE Labyrinth — 30 maze game £4.50 Chancellor — Run UK economy £4.50 Othello — A version of the popular boardgame in machine code £4.50 Disassembler — £4.50 Colossal Adventure — The original mainframe game £8.95 Adventure Quest £8.95 Dungeon Adventure — Try to collect the Demon Lord's treasure £8.95 Monster Mine £6.95 Sultan's Maze — 30 maze game £6.95 Golf — 18 hole course with handicaps £6.95 Games Vol. 1 (Tower of Rings, Mastermind, Blocker & Yahtzee) £4.50 BOOKS "Lynx Computing" 1983 I. Sinclair £5.95 "Z80 Instruction Handbook" 1982 Wadsworth £4.50 Parallel expansion bus connector + pushbotton to reset Lynx £4.50.

Lynx dust covers; high quality & washable £2.50

Lynx cassette leads DIN to DIN plus remote or DIN to 3 jacks £2.35

THIS IS JUST A SAMPLE — SEND S.A.E. FOR DETAILED CATALOGUE

Mail order with cheque/PO to:
FULCRUM PRODUCTS Dept. L
14 STEEP LANE, FINDON
WEST SUSSEX, BN14 0UF

SEND FOR FREE CATALOGUE