NILUG

NATIONAL INDEPENDENT LYNX USER GROUP

NEWS

Magazine for **LYNX** Users

Volume 1. Issue 1.

NILUG NEWS

VOLUME 1. ISSUE 1.

CONTENTS
========

EDITORIAL

I purchased a LYNX during April and started digging around inside. I made a few discoveries like Basic reserved words which are not in the manual. I wondered what else the LYNX had to offer. Since I own another well equipped micro I made a 2764 eprom reader and read the two LYNX eproms into my other little beast. Then I hit the code with a disassembler. About 180 pages of output later there it was in its glory. Now all I had to do was find out how it all worked.

I soon found the lists of Basic reserved words, the Monitor and a very interesting block of ram from 61EE to 62E0 which had a lot of pointers,jumps and data in it. Finding out how data was displayed on the screen was a little harder but I simply stuck at it and that is no longer a mystery. Explaining it may prove to be quite another matter. (See the article on output to the screen).

I then looked around to see if someone had set up a user group, after all I would like others to have the information I have gathered (provided that they give me what they have discovered). I couldn't find a group. I then thought about the newsletter that Camputers are going to publish but I'm sure you haven't received your copy yet either.

It seemed it was up to me to start the ball rolling and here it is. My aim for the group is to assimilate and disseminate information about the LYNX for the benefit of members. The main offering will be NILUG NEWS which will be published six times a year. Other publicat -ions may follow.

If you would like to see your name in print here's your big chance. You may feel that you can't write or draw diagrams. Don't worry about that. Ideas and what you have to say are far more important than how you say it. I'm sure I'll be able to add a bit of polish if necessary. All published material will be paid for so if you want some extra pocket money start writing.

I do intend to take advertising. This will, however, be strictly limited to a maximum of say 15% of the newsletter. Taking advertising has two advantages for members. Firstly it keeps down the cost of membership. Secondly it allows you to see what products are on sale for the LYNX without having to wade through pages of adverts for other machines. However if you don't agree with this policy then let me know.

Articles, reviews, subscriptions etc should be sent to the address inside the back cover.

R.B.Poate
EDITOR.

OUTPUT TO THE SCREEN

I am going to explain how characters are output to the screen by starting at memory location 0000 and processing the eproms as if you had just started your LYNX. The reason for this is that I have the impression that there may be different versions of the LYNX eproms in circulation. Consequently any addresses I give may not apply to your machine. I have written to Camputers about this matter but so far I have had no reply. By starting at 0000 and working forward it should enable you to examine your code should you wish to do so.

I am going to assume that you understand binary logic (ANDing and ORing) and a little about machine code. If you don't and you would like to then drop me a line. If there is sufficient demand I'll write a few articles. One final point before we start, all addresses and data are given in hex unless otherwise stated.

Starting at 0000 then, we find that interupts are disabled, 20 is output to port 80 and then a jump is made to 003B. Here we find a routine which initialises the 6845 VDU controller. At 0052 a jump is made to 168C.

At 168C a data table is copied, the stack pointer is set up, and a routine is called to initialise ram. At 16B9 the HL register pair is loaded with the address of a string of bytes to be output to the screen. The bytes are stored at 17ED and they look like this :-

04 07 18 F3 F4 F5 F6 F7 F8 F9 19 0A 0A 0D 00.

In fact they display the LYNX logo which appears on the screen when you start up.

Bytes to be output to the screen fall into 3 categories. Firstly there are the special bytes which are less than 1F. These perform functions like the 04 above which clears the screen. The 07 sounds the beep etc. If these numbers seem familiar its because they are the same as those used by the Basic VDU command.

The second type of byte is the normal character byte and is in the range 20 to 7F. The third type is the graphics character and is in the range E0 to F9. As you can see there are seven bytes in the sample above which fall into this group. They are F3 to F9 and they form the LYNX logo. One final point to note about the string of bytes is that it is ended with a null ie 00.

After HL is loaded with the address a routine at 3539 is called. This routine will output the string of bytes to the screen. In turn it calls a routine at 352F which outputs a single character to the screen. Now we call a "print/don't print" routine at 203C. This routine picks up a flag stored at 6202. If either bit 0 or bit 7 is set then output to the screen continues. Otherwise a return is made to the character output routine.

If printing is to continue an address is picked up from 6200 and it is loaded into the HL register pair. The BC register pair is loaded with 204F and pushed onto the stack to form a return address. Now a

JP (HL) is performed. Since 6200 held 06A4, in effect a CALL 06A4 has been performed.

The routine at 06A4 performs two functions. Firstly it handles the processing of bytes 01 and 02 (set INK or paper colour) and secondly it decides whether or not the byte to be output is one of the remaining special bytes or a normal/graphic character byte. So now a split is made.

Remaining Special Bytes.

If the byte is a special then a jump is made to 06CE. The code here picks an address out of a table located from 06EA to 0729 and then a jump is made to that routine.

The routine addresses are listed below beside their byte number. It should be noted that routines 00,01,02,03,11,1A and 1B are simple returns since these options are not implemented.

00 06CD	01 06CD	02 06CD	03 06CD
04 0762	05 080C	06 0787	07 092D
08 0820	09 07A2	0A 077E	0B 06CD
0C 0DC5	0D 07B7	0E 0828	0F 082E
10 0768	11 06CD	12 073C	13 076F
14 0734	15 0736	16 07DE	17 0765
18 072A	19 0730	1A 06CD	1B 06CD
1C 0810	1D 078B	1E 07C1	1F 077B

Normal and Graphics Characters.

Normal characters are in the range 20 to 7F and graphics characters are in the range E0 to F9. A call is made to 62B8 where the bytes C3 9A 00 may be found. Hence a call is made to 009A.

At 009A the routine decides whether or not the character is to be printed normal height or double height. It decides this by looking at the byte stored at 6273. Location 6273 contains either 20 for normal height or 40 for double height. I'll explain normal height characters first and then explain how double height characters are formed.

Normal height

A jump is made to 00C2. Here the main register set is pushed onto the stack and a call made to 00E7. At 00E7 a call is made to a routine at 00CE which will work out where the 10 bytes of data which will form the character are stored. Bit 7 of the A register (which holds the character) is tested to see if it is a normal character or a graphics character and the appropriate offset is picked up from either 626F or 6271. The routine then calculates where the 10 bytes of data are which will form the character. A return is made to 00EB. The cursor horizontal position is examined to see where on the screen the character is to be placed.

The Hardware

Now consideration must be given to the hardware. The screen is made up of 256 pixels across by 248 pixels down. The hardware for this arrangement is 32 bytes across (32 times 8 bits is 256) by 248 bytes down. In a sense then, the screen may be considered to consist of 32 8-bit columns. This is important because characters are 6 bits wide and consequently they may be placed entirely in one column or they may straddle two columns. In fact there are eight different horizon-tal placements for the character. It may start in any one of the eight bits of a column.

In the LYNX the situation has been simplified somewhat by the fact that the PRINT @ option deals with horizontal positions which are measured in 2 pixel columns. Hence on the LYNX there are only 4 positions to consider. See the diagram below.

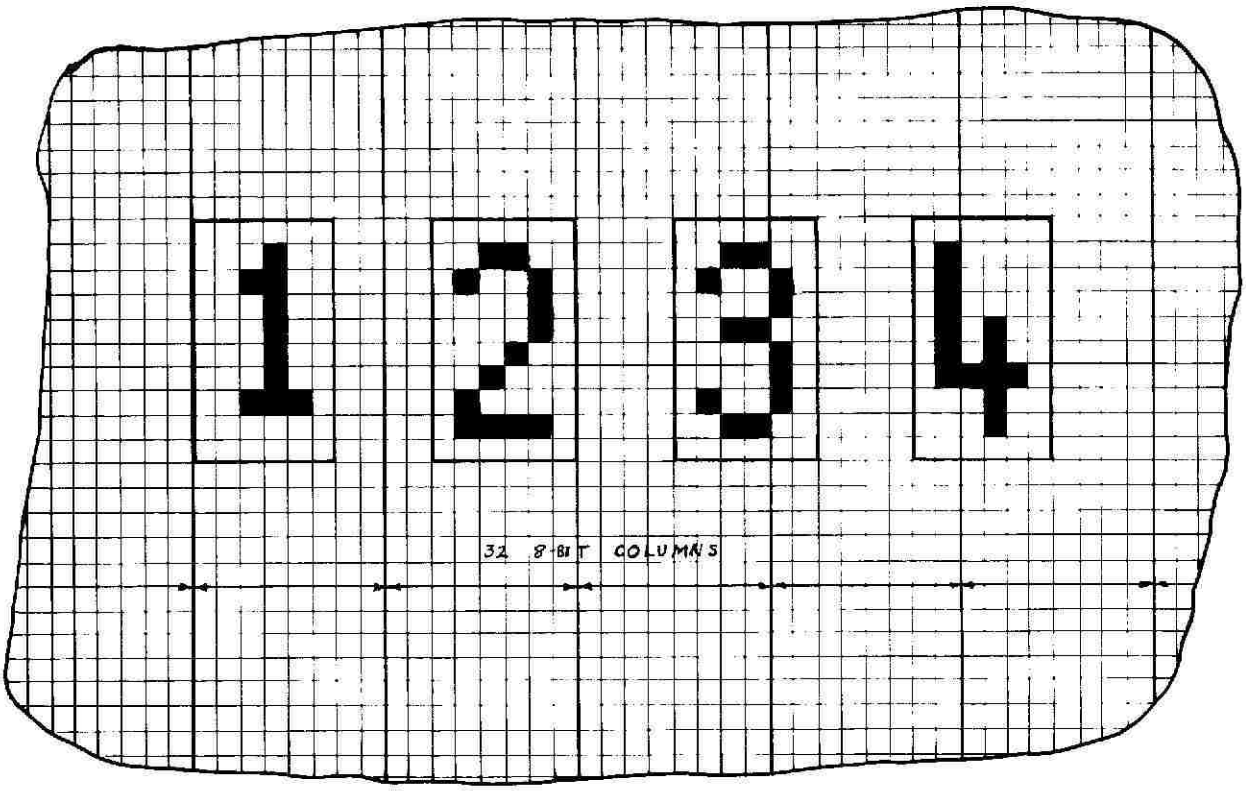


Diagram showing the four positions in which characters may be positioned.

Just to recap we are 6 routines deep, the 10 character bytes have been located and we have examined the cursor location to see in which of the four positions the character is going to be put.

Now a jump is made to one of four routines to take care of each of the four possible position options. Each routine performs the same function but each does it slightly differently because it is putting the character into a different position on the screen. I'll explain the routine for the first position. If you want to understand the others you'll have to work them out for yourself.

At this stage DE points to the position of the first of the 10 bytes which will form the character. HL points to the byte to be changed on an imaginary screen of 32 bytes by 248. (I've described it as an imaginary screen because later on an offset is added to HL to point to the position on the physical screen).

The routine sets up a loop to work down the screen for 10 rows to form the character. It then reads a byte from 6275 which is the overwrite on/off flag. This is either 00 for overwrite off or FF for overwrite on. This is ORed with 03 (0000 0011 binary). The result is loaded into the C register for safe keeping. The first of the character bytes is loaded into the A register. Two left rotates are performed to get the bits defining the character into the six most significant bit positions. Now the byte is ANDed with FC (1111 1100 binary). This has the effect of setting bits 0 and 1 to zero. So now the A register has the byte to be output to the screen and C has either FF or 03 to take care of overwrite.

A call is made to 626C which contains a jump to 085E. Here the ink and paper colour are picked up from 625B and 625C. The C register (overwrite on/off) is loaded into the E register and the A register (byte to be output) is loaded into the D register.

Now the routine works its way through each of the colours. A call is made to 0844 to decide if the ink, the paper or both require the colour BLUE. Upon returning the protection byte is picked up from 626B. If BLUE is protected then the next part is skipped. If BLUE is not protected then the A register is loaded with E8, the A' register is loaded with 63 and BC is loaded from location 628E which happens to contain 8000.

A call is made to a routine at 08B6. HL has the location of the character on the imaginary screen. HL has BC added to it to generate the physical position in memory. The byte 63 (A' register) is sent out to port 7F. The byte 40 is output to port 80 and then followed by the byte E8 (A register). Now the byte pointed to by the HL register pair is read into the A register. The A register is modified by ANDing it with the E register. The E register either contains FF or 03. By ANDing it with the byte which has been read off the screen the part of the byte that has nothing to do with our character has been left alone. In addition if overwrite is on ie E contains FF then all the bits which were set in the byte read from the screen will remain set. Next the byte is ORed with the D register. This has the effect of modifying the byte so that it forms part of the character we wish to display. After modification the byte is written back to where it came from. Zero is then sent out to port 80 and 7F. Now a return is made for the next colour.

The next colour is RED. If RED is not protected then the same bytes are loaded into registers A and A' but this time BC is loaded from 6290 which contains C000. A call is made to 08B6 as it was for BLUE. Finally, if its not protected, GREEN is done. The A register is loaded with E4, the A' register is loaded with 65 and BC is loaded from 6292 which contains C000.. The final call is made to 08B6.

Now the code returns to loop down the screen to put out the 10 bytes which form the character.

Do you see why the LYNX is a little on the slow side when it comes to displaying text ?

Double Height Characters

Double height characters are handled by outputting each of the 10 character bytes twice. Once in the normal position and once beneath it. This is achieved by using the normal routine for the first byte. Then the BC offset (normally C000 or 8000) is set to be either C020 or 8020. Then the normal routine is called a second time. Finally the offsets are reset to their original values. By increasing the BC offset by 20 (decimal 32) the second time the byte is displayed it appears on the screen beneath the first byte. Since all 10 bytes are displayed twice in this manner a double height character is formed.

Using it from Basic

How do you put data out to the screen from Basic ? The simple answer to that question is with difficulty. The main problem is that as soon as you switch in the video ram you switch out the program ram. If you try performing the various outputs to ports 7F and 80 using program ram the LYNX will crash. What happens is that the program counter suddenly goes from pointing at the real program to pointing at the video ram. The Z80 then picks up junk and a crash results.

So routines to use the video ram must be in eprom. This limits most LYNX users to the routines in the eproms already. If your eproms are the same as mine then you will be able to pick up the routines at the addresses I've detailed. If not then the best way is to pick up the routines by calling their ram pointers. As an example the routine pointed to by 626C could be used.

This routine needs the A and C registers set up plus the HL register pair. The A register has the byte to be put out to the screen. The C register has the overwrite on/off byte. The HL register pair points to a location on the screen. It should be in the range 0 to 2000. Here, then, is a simple demonstration program.

The machine code in line 40 looks like this :-

```
LD A,xx
LD C,xx
JP 626C
```

The 'xx' bytes don't matter since they get overwritten by the program.

```
10 CLS
20 REM Set up a machine code routine
30 REM it sets registers A and C
40 CODE 3E xx 0E xx C3 6C 62
50 REM find the position of the '3E' in line 40
60 LET A=LCTN(40)
70 REM randomise the byte to be output
80 POKE A+1,RAND(256)
90 REM randomise which bit should be put out
100 POKE A+3,RAND(256)
110 REM randomise the colour by changing the protection byte
120 POKE &626B,RAND(8)
110 REM call the routine with a random HL value.
130 CALL A,RAND(&2000)
140 GOTO 80
```

SUMMARY OF CODE

```
3539 Routine to put out a line of text.
: 352F routine to output a single byte.
: : 203C print/don't print routine.
: : : 6200 (06A4) Normal/special split
: : : : Normal characters
: : : : 62B8 (009A) single/double height characters
: : : : : 00E7 Locate location for character on screen
: : : : : : 00CE Load HL with the address of the 10 bytes
: : : : : Work out which of four routines to jump to
: : : : : For position 1 - 0130
: : : : : For position 2 - 014B
: : : : : For position 3 - 01B9 & 019E
: : : : : For position 4 - 0181 & 0164
: : : : : : 62C6 (085E) Work through the colours
: : : : : : :       BLUE     A=E8; A'=63; BC=8000
: : : : : : : 0844 Decide if ink and/or paper need blue
: : : : : : : 08B6 Output byte to screen.
: : : : : : :       RED      A=E8; A'=63; BC=C000
: : : : : : : 0844
: : : : : : : 08B6
: : : : : : :       GREEN    A=E4; A'=65; BC=C000
: : : : : : : 0844
: : : : : : : 08B6
: : : : : : return
: : : : : return
: : : : Special characters
: : : : Jump to special routine
: : : : return
: : : return
: : return
: Adjust cursor position
: Return for next character.
return to calling routine
```

USE THAT SPARE SOCKET

Perhaps you have have had the lid off your LYNX and seen the spare eprom socket sitting next to the two 2764 eproms. If you think about the memory map for a minute you will soon realise that there is a 'missing' 8K between the Basic/Monitor eproms and the ram. You don't need to be Sherlock Holmes to deduce that the spare eprom socket is wired into this 8K block.

Wouldn't it be nice if we could use this socket? Of course if you have access to an eprom blower which will handle 2764s then your lucky, but who has ? I own a blower which will blow 2716 eproms. Now there are quite a few blowers around which can blow these. If you don't own one then ask around at your local computer club. I'm sure you'll find one if you try. What do you mean you don't belong to a club? Join one!

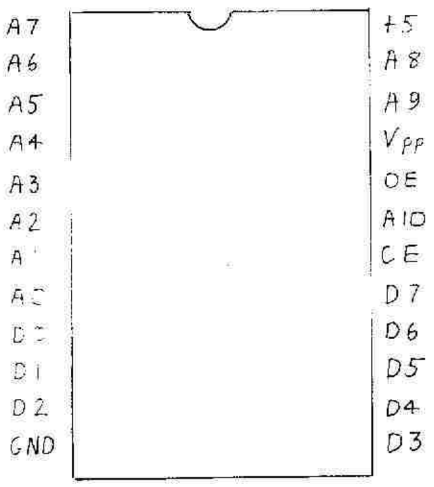
I found out that the Basic DISK command executes at 402D. This is in the spare eprom socket. This then gives a convenient way to execute any programs located in the eprom, just type DISK. Of course if the DISK command executes at 402D then it seems reasonable to assume that another chip is on its way to put in the socket when disks are added. However this doesn't matter. When I've got disks I wont need programs stored in eproms.

The idea then was to fit a 2716 eprom into the socket somehow. The first problem to be considered is that a 2764 has 28 pins whereas a 2716 has only 24. However if you look at the pin assignment diagrams opposite you will see that only two pins on the 2716 require attention (pins 21 and 24). So if a 28 pin socket is slightly modified by bending pins 23 and 26 out and up and then these are connected to pin 28 with a piece of wire, an adaptor for a 2716 is formed. Two safety measures are advisable. Firstly put blobs of solder on pins 1,2 and 27 of the socket. This will prevent you from putting the 2716 in the wrong holes. Secondly put a piece of tape under the high numbered pins of the socket. This is to prevent any connection between the bent up pins and the socket on the LYNX.

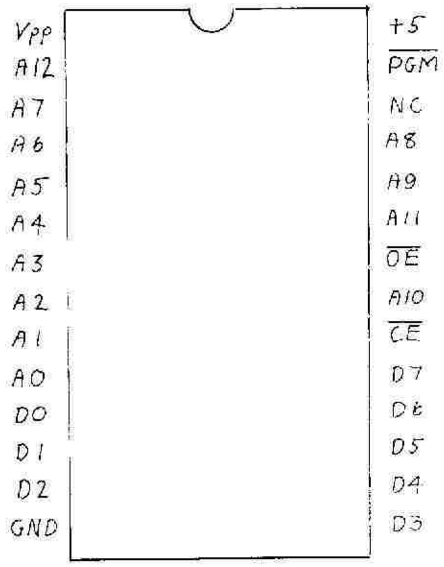
You may ask what are you going to put in the eprom ? Well my next project is to add a PIO (Parallel Input Output chip) to my LYNX. I will then be able to drive a parallel printer and link my LYNX up to my other machine. The eprom will hold the necessary code to initialise the PIO and set up an address at 62B8 to call the printer interface routines.

It was stated in the article on Output to the Screen that graphics routines must be in eprom. I have some ideas in that direction as well. Finally I would like to link my LYNX to my other micro, then I would be able to write code in assembler on the other machine and transfer it to the LYNX for running.

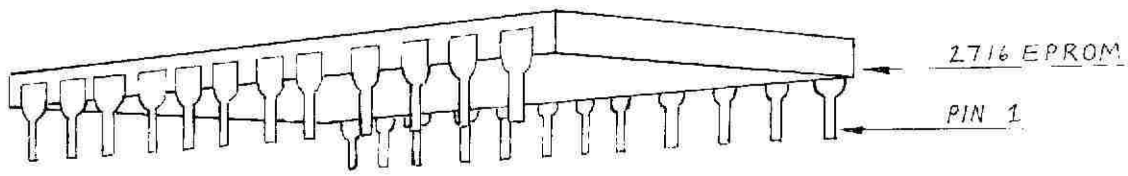
One final point. Be very careful how you put the adaptor into the socket and the 2716 into the adaptor. Its very easy to bend the pins and even easier to put the eprom in the wrong way round (I speak from experience).

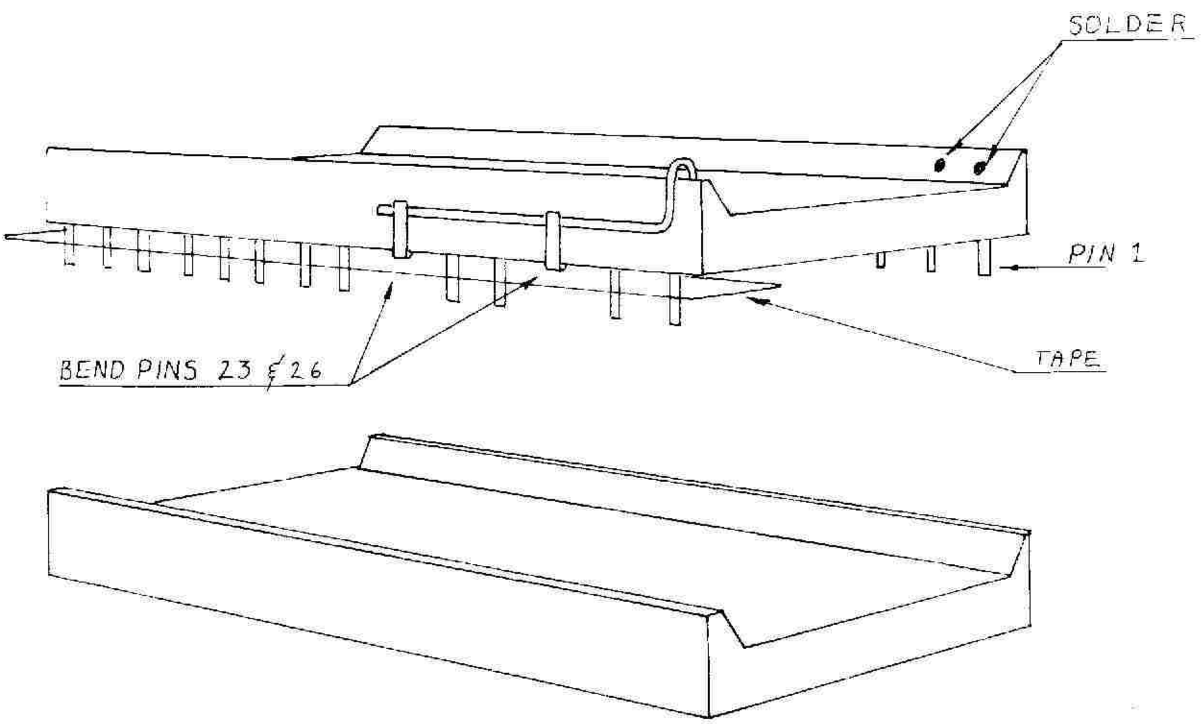


2716



2764





IC 44

USEFUL LOCATIONS

There is a block of memory running from 61EE to 62C0 which is initialised when the LYNX is switched on. It contains many useful addresses, pointers, and data bytes. I haven't worked them all out yet but those which I have are listed below. The layout takes the form :-

First address , End address , Data , Notes.

It should be noted that the data bytes are initialisation values and they may change. If anyone has any more information on this block of ram then please let me know and I will print an update in a subsequent issue.

61EE 61EF 00FF Stack pointer is saved here. Used by HIMEM routine.
61F0 61F0 01 Set up by RANDOM. This byte is used as a seed in the generation of random numbers.
61F4 61F5 0000 The storage for HL after a return from a machine code routine.
61F6 61F7 0536 Used by RUN to store a pointer to where the program has reached. The data '0536' points to a memory location which contains the byte 80. This may then be tested for continue / cannot continue condition.
61F8 61F9 4C69 A pointer to the byte before the start of the program.
61FA 61FB 4D69 A pointer to the start of the program.
61FC 61FD 4D69 A pointer to the end of the program.
6206 6206 01 This is used by LINK and as a print/don't print flag.
620F 6210 0767 Pointer to start of scalar variables ie A,B etc. In actual fact the variables are stored from 670C onwards. Scalar variables take five bytes each.
6213 6214 E468 Pointer to the start of a list of pointers to the string variables. The pointers take four bytes each. Only the first two bytes are used.
6215 6216 6813 A pointer to a list of execution addresses for a list of Basic words.
6217 6218 E111 A pointer to a list of Basic words.
6219 621A 1614 A pointer to a list of Basic words.
621B 621C 6015 A pointer to a list of routine addresses used for the validation of Basic words.
621D 621E F615 A pointer to a list of routine address for Basic words.
621F 6220 4E69 Address to the next free byte of memory after the program or the data.
6221 6223 C3 3C 27 EXT validation address.
6224 6226 C3 32 3B EXT execution address. This jumps to the routine which prints 'NOT IMPLEMENTED YET'.
6236 6236 00 Used to store bytes read from port 80.
6254 6254 03 Cursor horizontal position.
6255 6255 05 Cursor vertical position.
6256 6259 03 7B 05 F5 Window size.
625A 625A 00 Cursor on/off flag; 0 for cursor off, 1 for cursor on.
625B 625B 07 INK colour.
625C 625C 00 Paper colour.
625D 625E 0002 Cursor speed.
625F 6260 20EF Cursor characters.
6261 6261 00 This byte is used to flag SPEED and TRACE options.Bit 0 is

used by TRACE. Bit 1 is set if the SPEED option is in use. 62C1 is used as the SPEED store.
6262 6264 C3 BE OF Jump to a graphics routine.
6267 6268 0000 Graphics cursor.
626B 626B 00 PROTECT store.
626C 626E C3 5E 08 Call to character output routine.
626F 6270 9400 Offset for normal characters.
6271 6272 D401 Graphics characters offset.
6273 6274 20 00 Test byte for normal/double height characters. Hex 20 is for normal height, 40 is for double height characters.
6275 6275 00 Overwrite on/off. Set this byte to 00 for overwrite off, FF for overwrite on.

Now here are some execution addresses for Basic words which are not in the manual. I had hoped to print an article on how these extra words may be used but it turned into another heavy-weight article like OUTPUT TO SCREEN so it will have to wait for another issue.

6276 7628 C3 32 3B LIGHTPEN execution address. It is set to jump to a routine which will print 'NOT YET IMPLEMENTED'.
6279 627B C3 32 3B JOYSTK execution address. It is set to jump to a routine which will print 'NOT YET IMPLEMENTED'.
627C 627E C3 32 3B USER0 execution address. It is set to jump to a routine which will print 'NOT YET IMPLEMENTED'.
627F 6281 C3 32 3B USER1 execution address. It is set to jump to a routine which will print 'NOT YET IMPLEMENTED'.
6282 6284 C3 32 3B USER2 execution address. It is set to jump to a routine which will print 'NOT YET IMPLEMENTED'.
6285 6287 C3 32 3B USER3 execution address. It is set to jump to a routine which will print 'NOT YET IMPLEMENTED'.
6288 628A C9 00 00 Error trap address. This location is called before printing an error message.
628E 628F 0080 Offset for blue colour bank.
6290 6291 00C0 Offset for red colour bank.
6292 6293 00C0 Offset for green colour bank.
6294 6296 C3 32 3B Non maskable interupt jumps to this address. It is set to jump to a routine which prints 'NOT YET IMPLEMENTED'.
629A 629B EF38 Pointer to start of error messages.
629C 629F 5001 0403 Baud rate set up by the TAPE command. The data is loaded from area 0A0D 0DC4.
62A1 62A2 090D Pointer to data for SOUND.
62A3 62A5 C3 D4 0C A jump to a routine which will scan the keyboard for the '1' key. All other keys are disabled.
62B8 62BA C3 9A 00 Character output routine.
62BB 62BD C3 E2 10 Jump to line input routine.

TEXT

Have you found the word TEXT yet? Its to only word in the eproms which is not in the manual and which works. It sets the ink to green, the paper to black and then it clears the screen.

BREAK KEY

This is still a mystery to me. Has anyone found out how to use it ?

ERROR MESSAGES

There is a very nice feature of the LYNX Basic which impressed me when I found it. The error routine picks up a pointer to the list of error messages from 629A. It points to a null before the first error message. The pointer at 629A may be changed to point to a list of your own error messages. Ths could be used to make the LYNX ultra user-friendly.

In the short program which follows a list of error messages is set up which simply say "ERROR xx" (where xx is the error number). Its really a demonstration program rather than of any practical value. Doing it this way allows a lot of error messages to be set up very easily. In practice the messages should be a lot more helpful than those normally issued. As an example the seventh error message "SYNTAX ERROR" could be rewritten as "You have typed something I don't understand". To save space comments appear on the end of the lines in square brackets. Don't type them in!

```
100 REM ERROR yz  [ A data line. It gets zapped by lines 170 and 180]
110 LET P=LCTN(100) [A pointer to the E in ERROR]
120 LET L=&9000 [Where the new list is to be constructed]
130 DPOKE &629A,L [Set pointer to new list]
140 FOR I=1 TO 30 [Now set up 30 error messages]
150   POKE L,0 [The first null]
160   LET L=L+1
170   POKE P+6,&0030+INT(I/10)        [Overwrite the 'y' in line 100]
180   POKE P+7,&0030+10*FRAC(I/10) [Overwrite the 'z' in line 100]
190   FOR K=0 TO 7 [Now copy the new error message into the list]
200     POKE L,PEEK(P+K)
210     LET L=L+1
220   NEXT K
230 POKE L,0 [The last null]
```

RUN the program and then try typing ERROR 7 or 100/0 .

RESET/NMI (Non Maskable Interupt)

Have you ever crashed the system and had to switch off and start again ? My other micro has a very simple way out of this problem - you press the reset button. This starts execution at 0000 which is where the MONITOR is. You may then warm start Basic (a cold start is like the NEW command and you lose your program). Unfortunately, although the LYNX has the RESET line coming out the back it is not possible to use it because if the reset button were pressed the LYNX would cold start and you would lose everything.

There is, however, a Non Maskable Interupt line which could be used. The Z80 recognises three types of maskable interupts (ie it can be programmed to ignore them) and one NMI. The NMI executes at 0066. This jumps to 6294 which jumps to 3B32 and prints 'NOT YET IMPLEMENTED'. Location 6294 could be POKEd to point back into the Basic code. This would ,in effect, give the LYNX a reset facility. I'll write the code if someone will design the hardware. Any offers ?

NILUG

NATIONAL INDEPENDANT LYNX USER GROUP

53 KINGSWOOD AVENUE
SANDERSTEAD
SOUTH CROYDON
SURREY CR2 9DQ.

Editor: ROBERT POATE

The central aim for NILUG is to assimilate and disseminate information about the LYNX for the benefit of members. By acting as a clearing house for ideas and tips members will be able to get much more from their machines.

The main offering is NILUG NEWS which will be published six times a year. Members may submit articles, reviews, programs and letters for publication. The authors of the published articles will recieve payment for their efforts.

NILUG NEWS will take advertising although it will be limited to say 15% of the newsletter. Taking advertising has two advantages for members. Firstly if keeps the cost of membership down. Secondly it allows members to see what products are on the market for the LYNX without having to search through pages of adverts for other machines.

Subscriptions to NILUG are £9.00 per year (U.K.).

If you would like to join NILUG please complete the section below :-
Make cheques payable to NILUG.

= = = = PLEASE PRINT = = = =

NAME: -

ADDRESS: -

- -

- -

- -