TYNE & WEAR

ATARI 8-BIT

USER GROUP

Issue 26

March/April 1997

Published by TWAUG

Edited & Printed by AMG

Publishing

TWAUG NEWSLETTER is published bi-monthly, around mid-month of (Jan, Mar, May, July, Sept and Nov.)

It is printed and published by TWAUG, no other publishing company is involved.

Do you need to Contact anyone at TWAUG for a chat then phone

Atan Turnbull on: 01670 - 822492

or Max on: 0191 - 586 6795

Our Postal address:

TWAUG

c/o J.Matthewson

80 George Road

Wallsend, Tyne & Wear,

NE28 6BU

Max's Comments

It is difficult, when un-employed, with only the un-employment benefit coming into the house to keep up the subscription, we sincerely sympathize with anyone out of work, but why not be a writer.

You see being a writer would give you the chance to keep up with the issues. Write an article, on anything you have an interest in, or are familiar with and submit it to TWAUG. A contribution to the newsletter pays a free issue, or a choice of PD disks. We even supply the disks and postage should you so wish, for the article.

You could also, if you like, get any back issues you've missed, all you need to do is get your fingers dancing accross that keyboard.

Why not try your skill by writing for

T.W.A.U.G.

This appeal isn't only for the un-employed it is for anybody who's interested contributing towards the newsletter.

John's address

PUBLISHING!

This new look newsletter is set up with the Desktop Publishing program "TIMEWORKS 2", on the Mega 1 ST with 4 meg memory. Files are converted to ASCII and transferred to the ST with TARI-TALK. Those files are then imported into the DTP and printed with the Canon BJ-30 Bubble Jet Printer at 360 dpi, with excellent result.

TWAUG

NEEDS  YOU

TWAUG subscriptions

Home	1 Copy	£2.50
- DO -	6 Copies	£12.50
Europe	1 Copy	£2.50
- DO -	6 Copies	£13.50
Overseas	1 Copy	£3.50
-- DO --	6 Copies	£16.00

Issue 27 is due mid-May 97

ISSUE CONTENT

Letters

27th January 1997

Dear Max, John and Alan,

You ask how many subscribers use an Atari ST as a second computer. I do for one! It gets very little use in comparison with my 800XL but, if some ST articles appeared in the newsletter I might use it more. My 8-bit is quite ancient now, (aren't they all?), and there must come a time when it packs up and cannot be repaired. I am dreading that moment because I have a huge amount of disk files which I would hate to lose.

Porting them across to my ST would be a solution because, unlike the 8-bit, there are still numerous second-hand ST's on offer, and several firms repairing them. But I haven't the faintest idea of how to go about connecting the two computers.

I think we have the basis for a full-blown article here; somebody must have done the job so would they write about their experience,.....please!! Is the hardware/software still available? What price and where? Is there a d.i.y hardware kit and, once the two are connected, how difficult is it to getting them to respond.

Your publishing details mentions you use TARI-TALK for such transfers. It would be great if one of you could find the time to write of your experience with it, although I have been told that particular interface is no longer on the market.

I have one small criticism concerning the newsletter. Recent issues seem to contain mostly programming articles. I still read them but, considering the age of the 8-bit, feel that most of us have trodden that field before. I realise it can't be easy getting the right mix, especially if people are not sending you articles. Perhaps our American friends could be persuaded to part with some of their published articles. You have printed the odd one in the past.

Despite criticising I, for one, appreciate your dedicated, hard work. The new style newsletter is much easier on the eyes now using the bigger font, and improved layout. Thanks also must go to all contributors who, obviously, spend a lot of time and effort producing material to keep us, the readers, happy. Also, special thanks to John Foskett for his work on the Issue 25 disk. Together with all his previous contributions, he must have sent TWAUG a terrific amount of material. Although not a

Letters

games man myself, I do enjoy your demos, John.

Best wishes to TWAUG for the future.

Sincerely,

Dennis Fogerty.

P.S. The Textpro disk file of this letter has been saved on the enclosed disk. Should save you a bit of typing if you want to print part, or all, of this letter.

P.P.S. You often provide 'ARCed' files on your disks but I don't ever recollect seeing the documents for this utility. Also on the enclosed disk is the file SARC24.ARC which contains "Super Arc", "Super Un-Arc" and relevant documents (11 pages). I have taken SARC24.ARC from the A.I.M. disk April/May 1991. I have found the docs very useful - hope you do too.

Monday 17th February 1997

Dear Max,

In response to your "Survey", I enclose my subscription along with this letter... late due to operation after hand-injury! (Can't write/cycle/drive!!).

The 8-bit computer has followed us wherever we moved to... this home some 10 years ago! We've since gathered a family of three... who also use the 8-bit. We've also recently bought (cheap?) a (new) Jaguar (£40)... it seems a good game-machine - the children enjoy it. I'm not certain whether games are available... but it came with two games! We have many, many games for the 8-bit.

We bought a "Mega 1" some years back, and attached a hard drive. We have drive "B:" in 5¼ and 3½. The thought being that I could always consume surplus disks from the 8-bit... that drive has been desperately under-used! The Mega has since been upgraded to a Mega 4. The children like to play games on it too... though it has been used extensively for word-processing as the keys are easier to program for

Letters

complex chemical formulae!

The 8-bit takes less time to load "Textpro", and has also been used extensively for word-processing. It is attached to a daisy-wheel printer (bought cheap) as it prints directly to envelopes... not so with the dot-matrix attached to the Mega.

Due to the support of music we have remained loyal to Atari, though I have recently bought a PC (£20) for further word-processing. It is also possible that I'm going to get a 166 MMX for extensive graphics for a small business I'm running.

I've enjoyed staying with Atari, and won't be getting rid of any of the machines. I've gathered a small supply of surplus 8-bits so each of our children can have their own! It also suffices as an instant spares supply!

What a long-winded letter...

Terry R. Weizemann

Games for Jaguar are available, read "REPLY" section at the end of the "Letters".

Max,

TWAUG Survey

Re your query about articles on the Atari ST - I personally would not be interested.

I use a PC 386 DX40 in addition to my 800XL, and use SIO2PC to handle all my Atari files on the PC's hard disk - works like a charm!

All the best

Terry Chamberlain

27/1/97

Dear John & Max,

Enclosed cheque for 6 copies. Thank you for the hard work you are putting in, which the results justify.

I am at last, having a serious look at "Cracking the Code", and have dusted down my old assembler!

Keep battling,

Allan Doyley

Letters

Please find enclosed cheque for £12.50 for 6 issue sub.

With regards to your ST Survey I would be pleased if you did include ST articles in the newsletter as I feel that I am not getting much out of the newsletter particularly issue 25 which dealt mainly with programming. Also I recently picked up a 520 STFM for £30 and I am currently looking for a suitable printer, so articles on WP and DTP would be welcome.

Gordon Tully

Dear John

Please find enclosed a cheque & order form for the next six issues of newsletter.

Thank you "Collectively" for your efforts in the past & those to come, it is atways looked forward to, and the contents are most useful.

Thank you all again,

Atarily yours

Frank Atkin

Dear TWAUG,

Please find enclosed my subscription for the next six issues. In particular I am enjoying the BIT-WISE and DISPLAY LIST articles at the moment. As to whether to include ST articles in the magazine I would welcome it since I have an ST also, which I bought without any manuals. I would be interested to read any information about it although I still enjoy the 8-bit as well and don't see any problem with including articles about both computers in the magazine.

Yours

Brian Walker

Reply

WOW!!!

What a pleasant surprise, all that correspondence! I am simple over the moon, keep it up and keep sending those letters, at least I now know that you are enjoying reading our newsletter. Your mail can be critical or complementary they are all welco-

Reply

me. Our thanks to all for your reply.

And now I would like to reply to each letter individually.

The first reply is to Dennis Dogerty's letter.

There are a number of ways to port 8-bit material to the ST. Tari-Talk is an easy way to transfer Word Processor files, DOS files and Listings, but unfortunately Tari-Talk is no longer available from Page 6. We will try and find out what's available and let you know. The STXFormer is also a cheap way of porting 8-bit material to the ST. It is an emulator and it's a cable you plug into the back of the 1050 drive and into the printer port of the ST. When using Nul-Modem you also need the 850 Interface, or there is the Black Box that does the trick also, but these last two items cost money to purchase. The first two items are less expensive.

Terry Weizemann thinks his letter is long-winded, not at all. It is always nice to hear from our members what computers they got and what they are used for. 16/32 System is advertising the prices range from £25.00 to £60.00. The address is:

16/32 Systems

173 High Street

Strood, Kent, ME2 4TW

Tel: 01634 - 710 788

Dr.Terry Chamberlain isn't interested in ST material which is quite understantable not having an ST. The TWAUG newsletter is an 8-bit missive, first and formost and will remain so. I do not intend to fill the newsletter with ST material, the idea is to include article that will help our members to manage their machines better. According to the replies a number of our members would like to see ST material to help them with their problems.

I must thank Allan Doyley for the complimentary note and I am very pleased that you are having a serious look at "Cracking the Code". At least I know that all my typing it out wasn't for nothing.

Reply

I am sorry Gordon Tully that issue 25 wasn't particularly appealing to you, rest assured this issue sees the last of those articles. I had requested some feed back before I started those articles and none came, so I presumed that every body was satisfied with those articles.

The article for beginners was requested and so was the Display List article. The Binary Code articles I put in because at the time I needed some material to fill up the pages and I had nothing else on hand, and as I said above I had no feed back.

But when you read Brian Walker's letter you will see that the BIT-WISE and DISPLAY LIST articles are appreciated.

Frank Aitkin is also very satisfied with our newsletter, thank you so much for your note.

Your letter, Brian Walker, gave me a real buzz, knowing that there are still members, who enjoy the material I put in the newsletter. Even so these articles aren't new they are still very in interesting to read. Not every body likes to read that type of material but these articles, especially Display List comes in handy when you are playing about with demo programs. So thank you Brian for your letter and I will do my very best to include in future issues material to help you with your ST.

Thank you all again for your letters.

I am unable to include in this issue any material on the ST as it is too shorter notice.

Some of you mentioned what you would like to know, like Word-processing, DTP and porting from 8-bit to ST, others didn't point out exactly what they would like to read. After you've read this notice it would be nice if you could tell TWAUG the specific problem you're having.

If the members who have an ST but no manual, could you let me know exactly what you're not sure about. if you want I could start from the very beginning on how to use the ST.

I am waiting for your reply!

MAX

Coding Capers

MUSICAL REFERENCE

written by TOMO.

Hello there again, as you see I have got round to writing another subject for you after all! Actually, the subject I'm referring to this issue should have been part of "appendix B1: SOUND AND MUSIC" in my programming manual, but since it didn't quite get included, here it is for all to use.

On your Atari, you have a full 9 Octave musical scale, but little does everyone realize how to access it. It's all to do with location 53768; $D208. See my manual, page-126 for full details. When the Atari is powered-up or initially turned on the 64KHz clock is set as default, but this clock on it's own can only access about 7 octaves properly. The other 2 octaves can be accessed with a 15KHz clock, whilst a 3rd clock of 1.79MHz can be used to access all 9-octaves and at a higher level of tuning precision, although having the gain of power also has it's drawbacks.

The table of figures that I've included here in this article are for reference purposes mainly, and they show every one of the Atari' musical notes over all of it's 9-octaves and throughout each of it's 3 clocks.

Very quickly, the reason behind PAL AND NTSC tables is because the clock frequencies are fractionally different (shown at the base of each scale). You don't have to veer away from the more popular NTSC values since the actual difference to the human ear is fairly insignificant unless your a professional musician. If in the case you are a pro. musician then you might find fault in the temperament of my scale. Note, it's not set at the international standard or the concert pitch. It's actually slightly above international, but don't let me lead you to think that POKEY's strings wouldn't take the stretch! If someone wants to improve on it then please go ahead!

In addition, you'll notice that I've supplied the fractional part (to the first digit) of each frequency value in the 15KHz and 64KHz clocks. This is simply there to show you how near (or far) the Atari pitch is from the aimed pitch. Anyway, since you can only use the integer value as the frequency, you should realize that frequencies with the same integer value are there simply to allow you to make your own mind up concerning which note this integer belongs too.

Coding Capers

Have fun!

Oh, as a last point. The clock frequencies were all calculated via the given facts of the crystal oscillations (listed below) given to me in kindness by Terry Chamberlain, thanks for all your help Terry! The 2 main (fastest) CPU clocks are easy to achieve by dividing the CPU crystal by 2, however, the 64KHz and 15KHz clocks are achieved by 2 division ratios, 56 and 228. Divide by 56 to obtain the 64KHz clock and 228 for the 15KHz clock! Simple when you know how isn't it!

CRYSTALS:

PAL-CLR-CLK = 4.433618MHz "

CPU-CLK = 3.546894MHz / 2 = 1.77345MHz

NTSC-CLR-CLK = 3.579545MHz "

CPU-CLK = 3.579545MHz / 2 = 1.78977MHz

Until next time, have fun.

MULTIPLE FONT OUTPUT ON THE 1029

written by TomoHawk.

Although the Atari 1029 printer isn't very much of a qualitable output compared to many other printers, it still is used by some I'm sure, so I've sent you a program that will print your desired files, programs or text on the 1029 in any computer font you like. All you have to do is run the program and when option-D is selected to use a computer font, just insert a disk containing *.FNT files. Just use the OPTION and SELECT keys to pick the font file of your choice, press START and the font will be converted into a 1029 font during load.

In actual fact, 1029 fonts are only 7-rows deep, so 1-row of the computer font has to be lost. The current condition of my program gets rid of the bottom row of a computer font character. If you wanted to get rid of a different row then you can sort it all out by changing the successive numbers in the array 'NA' on lines 280-292.

The actual characters of a computer font are uncoded from their normal horizontal binary sums and recoded into vertical sums. For example;

Coding Capers

	128	64	32	16	8	4	2	1
1	0	0	0	0	0	0	0	0
2	0	0	0	X	X	0	0	0
3	0	0	X	X	X	X	0	0
4	0	X	X	0	0	X	X	0
5	0	X	X	X	X	X	X	0
6	0	X	X	0	0	X	X	0
7	0	X	X	0	0	X	X	0
8	0	0	0	0	0	0	0	0

If the above X's denoted the bits set in a computer font character, then the binary sum for row-1 would be 0, row-2 16+8 = 24 and so on... But the same character on the 1029 would have to be uncoded from these sums to the pattern above, and then recoded horizontally like so:

	c-1	c-2	c-3	c-4	c-5	c-6	c-7	c-8
128	---------no such row---------							
64	0	0	0	0	0	0	0	0
32	0	0	0	X	X	0	0	0
16	0	0	X	X	X	X	0	0
8	0	X	X	0	0	X	X	0
4	0	X	X	X	X	X	X	0
2	0	X	X	0	0	X	X	0
1	0	X	X	0	0	X	X	0

Instead of coding them vertically, they would now get coded horizontally, so col-1 would be 0, col-2 8+4+2+1 = 15 etc..

Anyway, I'm sure you follow. Hope this piece of text and program comes in useful for you. Happy computing.

1029 Utilities

Looking for more 1029 printer utilities? There is a double sided disk of just such utilities in the TWAUG's library.

This disk is crammed full of goodies:
Picture Dump program

Poster printer program

Print Shop utility program

Screen Dump, List 1029, Label Printer, and much more.

Library Number TW433

There is another very good program available from MICRO DISCOUNT

it's called PRINT-WORKS, it is a Document Processor for the Atari 1029 printer.

'Hi' Guys.

I have been meaning to contact you for a long time, but being short of money I have not been able to send of for a subcription. John Foskett from Kingston told me that you were short of things to publish, and I hope these few things will help you.

Find enclosed disk Side B (boot with basic) an article on making power supply units for the Atari 8-Bit Keyboard, which you may see fit to publish as an article or include in your P.D. library, if you have one it's up to you. Also on Side B of the disk you will find all the documents, and a program from Analog Mag which will print out the circuits for you, on an Epson or compatible printer. I don't know if you like hardware articles, but it could help somebody out there who understands the basics of electronics.

On Side A of the disk you will find a National Lottery Calculator program I wrote which you may see fit to use also a Program which will dump Colour 62 sector picture files to a Citizen ABC colour printer in full colour. The colours which you can select from within the program before dumping to the printer by pressing keys 1 to 4 on the keyboard to select the colours you wish your printer to print. I don't know if this program will work with any other make of colour printer, but I should think it will. I also have put on Side A of the disk a list of Epson printer codes (Mini Office II file) a Doc file which will print out in colour the code's which controls Citizen printer for large text, wide text, colour printing, ect., these printer codes should also work OK with other Epson compatibles. Side A also covers a project on building an Epson or compatible printer interface which you may find a use for. Read all documents before use. In fact you will have to print them out to read them properly. Also a doc file for the Atari Assembler Cartridge and a doc file for MS-DOS commands Versions 3 to 5

Yours sincerely,

M.Tomlin W.A.C.O.

SALE

Atari 800XL Computer, 1050 Drive, 850 Interface, all leads and power units, 2 joysticks over 100 games and Star-LC10 printer

£70 or near offer

contact: G. Lewsley

225 Homethorpe, Hall Road

Hull HU6 9HP

Tel: 01482 - 856 179

GRAPHICS - DISPLAY LIST

MIKE ROWE concludes his series on how to produce brighter displays

THIS issue's article, the last in the series, takes a look at some non-standard graphics modes and rounds off with non-standard display.

Right at the beginning I said 16 modes were available to the Atari user, but this can be stretched by a further 12 modes when you include text windows where available.

In reality things are not this simple. These 28 modes are only those directly available using the operating system on XL and XE models only.

Graphics modes 12-15 are available on the 400 and 800 but only by creating your own display list as demonstrated in the second article in the series.

In reality it is possible to get many more modes than this -would you believe over 100 different graphics modes?

Firstly there is another Antic mode which is not supported directly by any of the current machines. This is Antic Mode 3, which can be obtained easily by creating your own display list.

It is essentially similar to Graphics 0 but with one difference - it allows true descenders. That is, the tail in the small y comes properly below the rest of the letter.

This is because it interprets the data for the character differently. A normal character is 8 pixels wide by 8 lines deep. In Antic Mode 3 it is 10 lines deep and the two bottom scan lines appear blank. In addition some characters, notably lower case as well as a few others, are displayed with the first two bytes of the character appearing at the bottom of the character.

As you might imagine, the standard character set would not be suitable

Going out in a blaze

GRAPHICS - DISPLAY LIST

for this mode. You really need a custom set.

In the example in Demo 1 I have used the internal set for briefness. However, I have offset the character set one byte lower and moved the lower case set's last bytes to the first bytes. This gives a workable version of the character set.

In addition, to show the true lower case, I have redefined some of the characters to give true descenders.

Secondly several useful modes are based on Graphics 9, 10, 11. The first

of graphics glory

Ataris produced did not have Graphics 9 - 11.

Before late 1979 the computer had a chip called CTIA, which provided Graphics 0 to 8 only. After this they fitted the GTIA chip allowing the three new modes.

In fact the display list is exactly the same for these modes as it is for Graphics 8.

The secret of the difference lies in memory location 623. Numbers from 0 to 15 are poked here in the use of player-missile graphics to decide priority - that is, which player shows in front or behind what.

However numbers from 64 up - bits 6 and 7 - will enable the GTIA modes.

If in Graphics 8 you POKE 623,64 (bit 6) you get Graphics 9. POKE 623,128 (bit 7) gives graphics 10, POKE 623,192 (bits 6 and 7) gives Graphics 11.

Leading on logically from this, the same could be done in any mode. This gives a theoretical maximum of 52 full screen modes and a further 52 modes with text windows. That is a staggering 104 graphics modes.

Don't det too excited. This is indeed possible, but most of them are quite useless, some are identical to others and all the text windows are illegible.

The last point can be circum-navigated and will be dealt with later.

Probably three new modes are definitely usable and significantly different. These are shown in Demos

GRAPHICS - DISPLAY LIST

II and III.

Demo II is a 7 colour 80x96 mode which takes only 4k of memory. Essentially it is a cross between graphics 7 and Graphics 10. For some reason you can only get seven out of eight colours of Graphics 10. Notice also that the colour registers used are not 0-7 as expected but are as in Table I.

Apart from this, the mode is just like Graphics 10 but with half the vertical resolution and half the memory usage. Similar hybrid modes can be used with Graphics 9 and 11 but are probably less useful.

Demo III is a seven colour 20x12 text mode which is a cross between Graphics 2 and Graphics 10 (by using Graphics 1 a seven colour 20x24 mode can be obtained). Again, because of the way the operating system works, seven colours as above are available. Also, as in Graphics 2 proper, only 64 different characters can be displayed at once - characters 32 to 95 -space to Z.

If the other characters are printed they appear as an allowable character but in a different combination of colours. This is much the same as Graphics 2 and is how the different colours are obtained.

However this cannot explain the availability of seven colours. This occurs because of the way the character set data is interpreted. A normal character is lit pixel by pixel controlled by eight bits giving a horizontal resolution of eight per character. This new mode is more like Graphics 12 (Antic 4) in that the eight bits give a horizontal resolution of four per character - that is, each

Colour register location	Colour numbers to use	Colour displayed
704	0, 1, 4, 5	0 (backgroung)
705	2, 6	2
706	3, 7	3
707	-	-
708	9	9
709	10	10
710	11	11
711	-	-
712	8, 12, 13, 14, 15	8

GRAPHICS - DISPLAY LIST

pixel of the character is controlled by two bits allowing control over the colour of each pixel. The character is therefore laid out as in Figure VI.

In graphics 12 this gives four colours (five with inverse) but in addition to the other method of colour selection mentioned before seven colours become available in the new mode. Therefore a custom character set is essential.

Due to the peculiarities of this mode, normal capital letters do not show up. Lower case and inverse will print the character in different colours as will printing characters 0-31 and 96-127 normally and in inverse. This is not straightforward in the way it occurs, and is best discovered by experimentation.

All the GTIA modes interpret character set data like this and this is why the text windows are illegible. A text window is easily obtained, however, by using Display List Interrupt to change back from the GTIA mode at the text window. This is shown in Demo IV, but the principle will work with any GTIA mode.

Some of the examples above may be difficult to grasp at first, especially III, but if studied carefully they are reasonably straightforward. Feel free to experiment with the programs to discover more.

Finally, to illustrate the power of the display list, I'd like to answer a problem posed by a reader. He wants a display comprising one row of Mode 2, 112 rows of Mode 15 and eight rows of Mode 0.

Although quite possible this is far from the easiest combination of screen modes. Firstly he has based his screen on an 8k mode - Graphics 15.

You may remember I mentioned any screen display crossing a 4k boundary needs a new load memory scan instruction in the display list where the 4k boundary is crossed.

In the 8k modes this therefore means that the list of mode numbers is interrupted half way down by three numbers.

The first is the mode number - say 14 for Basic mode 15 - + 64. This tells the operating system that the next two numbers are the low and high bytes of the screen memory after this point, that is the points to the next 4k block of screen memory.

If you now interfere with the display list above this the screen memory may well no longer remain consecutive at this point.

The second problem lies in the

GRAPHICS - DISPLAY LIST

decision to have a Graphics 2 line at the top of the display. This obviously causes problems as above. However in addition this mode requires only 20 bytes of memory per line. Graphics 15 requires 40 bytes per line.

As the OS expects 40 bytes per line everything below the Graphics 2 line will be offset by half the screen. Also the second 4k block of screen memory will be 20 bytes out of alignment with the first 4k of screen memory.

So much for the problems. Now the solution!

Well there are many solutions really but I think the easiest and probably shortest is shown in Demo V.

Here I have considered each of the three modes as individual screens. I started with a Graphics 15 full screen display, changed the top line to Graphics 2 and kept a track of the location of the start of screen memory for this line in LO1 and HI1.

I then inserted a new load memory scan instruction (LMS) and offset the screen memory for this by 120 bytes. This is to avoid the necessity for moving the location of the later LMS which is there to cope with the 4k boundary which is crossed by Graphics 15.

I again kept track of the start of this block of screen memory in LO2 and HI2. Finally after the requisite number of Graphics 15 lines I again inserted an LMS for the eight Graphics 0 lines. The display list is ended straight after this.

Now we have the display needed to treat each part as a seperate screen or possibly as a sort of window. This means as well as poking the mode of the area of screen we are using into location 87, we must also poke the start of memory fot that block of screen into 88 and 89.

The easiest way to do this is as a set of subroutines to be called. This will also mean that each block starts at location 0,0, thus avoiding printing to position 117,4 which could otherwise occur. The OS would not allow this in Graphics 0.

Phew - glad I got that off my chest. I think I'll take a break now.

THE END.

BIT WISE

MIKE BIBBY continues his series on binary numbers

IN this series so far we have learnt a lot about the binary system - the numbers our micro works in.

We have seen that its memory is divided up into bytes - a set of eight two-state, binary units called bits. Each bit can have the value 1 or 0.

If a bit has the value 1 we say it is set. If a bit has the value 0 we say it is clear.

As we're dealing with eight bits at a time, we can use various combinations of the bits in a byte to code any whole number (integer) in the range 0 to 255.

To do this we associate a code number with each bit. Figure I shows the scheme.

Our eight bits are labelled b7...b0 and the numbers associated with each number are shown above each bit. (The more mathematical among you will see that they're in ascending powers of two.)

To discover the value coded in a byte we simply add the numbers associated with every bit that is set (1), ignoring all clear bits (0). So:

%10101000

codes the number:

128 + 32 + 8 = 168

We also learned to do tricks with, or to put it more properly, manipulate, binary numbers. We could create the complement of a number - a sort of binary opposite - by changing every clear bit to set ("setting" the bit) and changing every set bit to clear ("clearing" the bit).

So the complement of the number:

%10101000

gives us:

%01010111

We can add and subtract binary numbers, as well as multiply and divide. We learned other ways of combining them too, with the logical operators AND, OR, EOR.

EOR, which stands for Exclusive OR, is also called XOR.

When combining two binary numbers under the influence of these operators we compare each bit in one number with the corresponding bit of the other.

Then, according to a rule which depends on the operator we're using,

BIT WISE

we decide whether that particular bit (the result bit) in the "answer" byte is set or clear. Table I shows the rules for the operators.

As we've said, a micro's memory is divided into byte-size compartments, called memory locations. Each location has a number associated with it so we know which one we're talking about.

These numbers are known as memory addresses.

Much of what a microprocessor does involves moving information - in the form of binary numbers - from one location to another.

If you cast your mind back to earlier articles, I said that each bit was like a switch - its two values 1 and 0 could be used to signify that the switch was on or off respectively.

Imagine that we could wire up one of our bits to a machine's on/off switch. Then by setting that bit we could switch the machine on, and by clearing it we could switch it off.

This sort of thing is possible, though we'd need to use some clever electronics. In fact, since we deal with eight bits at a time, we could arrange things so that a single byte controlled

AND	Sets the result bit only if both bits compared are set, otherwise the result bit is clear.
OR	Sets the result bit if either or both the bits compa- red are set. Only if both bits compared are clear is the result bit clear.
EOR	Sets the result bit if the bits being compared dif- fer in value. If the EOR bits compared are identi- cal, the result bit is cleared.

Table I: Rules for logical operators

128	64	32	16	8	4	2	1
b7	b6	b5	b4	b3	b2	b1	b0

Figure I: Values associated with bit positions

BIT WISE

the on/off status of eight separate machines -each machine m7, m6 ... m0 corresponding to an individual bit of that byte, b7, b6 ... b0. We'll term that byte the control byte.

We call such arrangements memory-mapped output, since what we put in memory maps, or sets the pattern for, what happens in the outside world. Most microprocessors support this or some similar sort of output. Figure II shows the type of scheme we mean.

The Masked bytes are taking control

Assuming we've got things connected up properly, if we then load the control byte with:

%11111111

all the machines would be on. Remember that if a bit is set the corresponding machine is on. If we want to switch all the machines off, we can load the control byte with:

%00000000

And, of course, we can have any on/off pattern of machines, setting or clearing the relevant bits by loading the control byte with new numbers. Loading it with:

%11110000

is one way of switching off half the machines.

Sometimes, though, we might want to switch a particular machine or two on or off without knowing (or caring) whether the others are on or off. This means we need some way of affecting only the bits controlling those machines, while leaving the others unchanged.

Suppose we wanted to switch off a machine - say m6. We can do this by making b6 of the control byte zero.

To clear that one bit to zero we AND the control byte with another byte - called the mask - the bits of which are set (1) except for b6, which will be zero. That is, we AND the control byte with:

%10111111

We then make this result our new control byte, and off the machine

BIT WISE

goes.

To see how it works in practice, let's assume that initially all the machines are on, so the control byte is:

%11111111

To switch machine m6 off we must AND it with:

%10111111

The sum is:

```
      %11111111 control byte
AND   %10111111 mask
      %10111111 new control byte
```

As you can see, the outcome is that when we update the control byte with the result, m6 is switched off while the others remain on.

The trick isn't hard to see. Let's consider things from the point of view of bits in the mask. If the bit is a 1, when you AND it with the relevant control bit the resulting bit is the same as the control bit. That is, ANDing a bit with 1 leaves that bit unchanged.

Think about it. If the control bit were 1, then as 1 AND 1=1, you're with 1. The bit's unchanged.

If, on the other hand, the control bit were 0 then, as 0 AND 1=0, the bit remains unchanged as 0.

In other words bits in the mask with 1 in them leave the corresponding control bit unchanged.

So for machines whose on/off status we don't want to alter -we may not even know if they're on or off - we set the corresponding bit in the mask to 1.

However if the bit in the mask were clear (0) it wouldn't matter what the state of the original control bit was - the result would still be 0.

Say the control bit was 1, then as 1 AND 0=0 the resulting bit is a 0.

Alternatively, if it were 0, since 0 AND 0=0 the resulting bit is again 0.

So bits in the mask with 0 in them set the corresponding bits in them set the corresponding bits in the result byte to 0.

This means to switch specific machines off we construct a mask consisting of 1s for the machines we wish to leave unchanged and 0s for the machines we want off - in the appropriate bit positions.

We then AND the mask with the control byte and then make the resulting byte the new control byte.

BIT WISE

Fine, but how do we switch on specific machines?

Well, we update the control byte by ORing it with another mask. This time we put 1 in the bits corresponding to the machine we want on, and 0 in the bits corresponding to the machines whose on/off status we wish to leave unchanged.

This works, since when you OR a bit (whether 0 or 1) with another bit whose value is 1, the answer is 1.

That is **0 OR 1=1** and **1 OR 1=1.**

So using a 1 in the relevant bit of an OR mask will set the corresponding result bit. When this becomes the new control byte the corresponding machine will be turned or left on.

On the other hand, ORing a bit in the control byte (no matter what value) with 0 leaves that bit totally unchanged since **1 OR 0=1** and **0 OR 0=0.**

So when we OR the bits of the mask that are 0 leave the corresponding bits of the control byte unchanged.

This means, to switch specific machines on we use a mask consisting of 0s for the machines we wish to leave unchanged, and 1s for the machines we want on - in the appropriate bit positions.

We then OR that mask with the control byte and make the resulting byte the new control byte.

Hence, to ensure that m6 is definitely on, we OR the control byte with:

%01000000

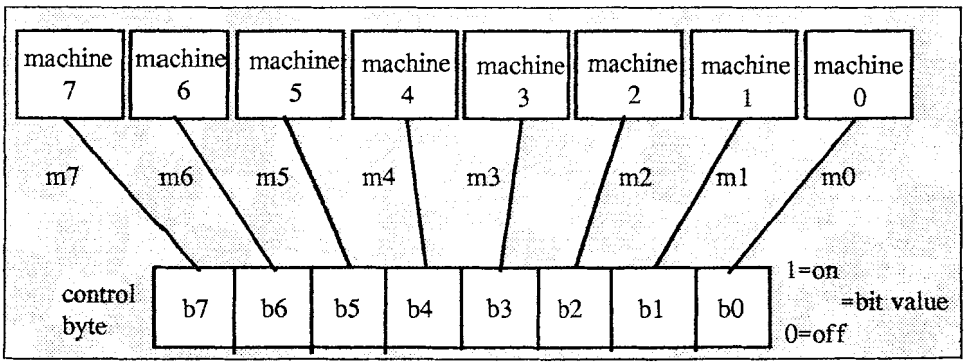


Figure II: Memory mapped control

BIT WISE

For example, if m6 is off, and all the rest on, to switch m6 on we do the following:

```
      %10111111  control byte
AND   %01000000  mask
      %11111111  New control
                 byte
```

Of course, both AND and OR have uses for the micro enthusiast other than controlling machines.

To illustrate one, consider the Ascii character set. The codes for A to Z are in the range 65-90, while their lower case equivalents, a to z, are in the range 97-122.

Looked at in this decimal way, there seems little relation between the upper and lower case sets. If we look at them in hex, though, we can see that:

A...Z runs from & 41 to & 5A

a...z runs from & 61 to & 7A

I hope you can see the pattern.

In fact the numerical Ascii difference between a lower case character and its upper case equivalent is always & 20. Looked at in binary, this difference is **%00100000.** In other words, bit five is set for lower case, and is clear for upper case - remember, we start with the zero bit.

For example, the code for A is:

%01000001

whereas the code for a is:

%01100001

Similarly, the code for Z is:

%01011010

and the code for z is:

%01111010

In both cases the only difference is in bit five.

So if we have an Ascii code for a letter, we can force it to be upper case by clearing bit five to zero. We can do this by ANDing the code for the letter with the mask %11011111 (& DF).

Remember, the bits in the mask that contain 1 will leave the corresponding bits in the Ascii code for the letter unchanged in the resultant byte, whether they be 0 or 1. On the other hand, the bit in the mask with 0 in it will force the matching result bit to be zero.

So:

BIT WISE

```
      %01100001 ( the code for a)
AND   %11011111 ( the mask-&DF)
gives %01000001 (the code for A)
```

It won't surprise you to learn that we can reverse the procedure - forcing upper case into lower case - by using OR to set bit five. This time the mask will be %00100000, the 0s leaving things unchanged in the resultant byte, the 1 forcing a corresponding 1 in bit five of the result bit.

So:

```
      %01011010 (the code for Z)
OR    %00100000 (the mask-&20)
gives %01111010 (the code for z)
```

One further use for AND is to test if a particular bit in a byte is set. We just AND that byte with a mask consisting of a 1 in the bit being tested, with 0s in all the rest. The bits with 0 in them, of course, set the corresponding bits in the resultant byte to zero.

Since the rest of the bits are already cleared to zero by the mask, the only thing that could stop the entire resultant byte being zero is the value derived from the bit under investigation:

- If that bit is set, the corresponding result bit will be set also (1 AND 1=1) so the resultant byte will be non-zero.
- If the bit being checked is clear, the corresponding result bit will be clear (0 AND 1=0) so the resultant byte is zero.

In machine code we can differentiate between zero and non-zero bytes fairly easily.

Let's see how this works in practice. If we were testing for bit four being set, the mask would be %00010000.

Try ANDing this value with %00110100, where bit four is set, and also with %00101100, where bit four is clear, and you'll see that the resulting bytes are non-zero and zero respectively.

So what of EOR/XOR? Well, its function is to return a 1 if the pair of bits being combined differ, and 0 if they're identical. Given this, we can use XOR to test which bits in a byte differ. For example:

```
      %10101110
XOR   %11001101
gives %01100011
```

BIT WISE

where the set bits neatly mark out the different pairs.

We can also use EOR/XOR to complement or NOT a byte, by EORing it with a mask of %11111111. Since the mask is all 1s, the result depends entirely on what's in the byte under investigation. Bits that contain 1s will give 0 (since 1 EOR 1=0), while bits that contain zero will give 1, since 0 EOR 1=1.

This is exactly what we want to happen with a NOT - change the 0s to 1s and vice versa. For example:

```
      %10101101
XOR   %11111111
gives %01010010 (the complement)
```

We can also use EOR/XOR to test if two bytes are identical. If the result when we EOR is zero, they must have been identical since every pair of bits must have given zero, which only happens when the bit values are the same.

If there's a non-zero result there must have been a pair of bits that differ, so the two bytes under consideration must differ. For example:

```
      %10101010
XOR   %10101010
gives %00000000
```

whereas:

```
      %10101110
XOR   %10101010
gives %00000100
```

which is, of course, non-zero, since the bytes differ.

We've probably already mentioned the use of EOR in graphics application programs where it's widely used for its "hey presto" effect. This is based on the fact that if you EOR a first byte with a second and then EOR the result of that once more with the second byte, the first byte reappears. Look at this, if you don't believe me:

```
      %01011100 (first byte)
XOR   %01110010 (second byte)
      %00101110 (result)
XOR   %01110010 (second byte
                 again)
      %01011100 (first byte
```

BITWISE

ATARI ASSEMBLER EDITOR CARTRIDGE

We use this EORing technique to draw things on a background and then move on, leaving the background unchanged. In this case the first byte is the background colour number. If we then EOR our second byte - corresponding to the colour number of whatever it is we're drawing - on to the background, it will be displayed in the resultant colour number. It's rather like mixing colours mathematically.

To get rid of what we've drawn, we draw it again with the same colour number, once more under the influence of EOR. Of course EORing twice with the same byte gives us the original byte back. This results in whatever it is being drawn appearing in the original background colour. Hey presto - it's gone.

Well, that's the end of the series. Hopefully you'll have gained some idea of the power of binary numbers and the ways they can be combined. I've only touched on a fraction of the potential uses, but you'll be well equipped to work things out for yourself from now on.

This concludes MIKE BIBBY's series on binary numbers

Summary of All Commands Using the EDITOR

NEW Command:
This command clears the edit text buffer. After this command you cannot restore your source program; it has been destroyed.
DEL Command:
This command deletes statements from your source program.
DELxx RETURN:
Increment statement number by 10 after each RETURN. The new statement number, followed by a space, is automatically displayed.
NUMnn RETURN:
Has the same effect as NUM, but the increment is nn instead of 10.
NUMmm,nn RETURN:
Forces the next statement number to be mm and the increment to be nn.
RETURN:
RETURN Cancels NUM Command.
REN Command:
This command renumbers statements in your source program.
REN RETURN:
Renumbers all the statements in increments of 10, starting with 10.
RENnn RETURN:

ATARI ASSEMBLER EDITOR CARTRIDGE

Renumbers all the statements in increments of nn, starting with 10
RENmm,nn RETURN:
Renumbers all the statements in increments of nn, starting with mm.

FIND Command:
FIND/SOUGHT/, RETURN:
Finds the first occurrence of the string SOUGHT. The statement containing such occurances are displayed.
FIND/SOUGHT/,A RETURN:
Finds all occurances of the string SOUGHT. All statements containing such occurrences are displayed.
FIND/SOUGHT/xx,yy,A RETURN:
Finds all occurrences of the string SOUGHT between statement number xx and yy. All the statements that contain the string are displayed.
In these examples, the string SOUGHT is delimited (marked off) by the character /. Actually any character except space, tab and RETURN can be used as the delimiter.
For example, the command:- FIND DAD finds the first occurance of the character A. The delimiter is defined as the first character (not counting space or tab) after the keyword FIND. This feature is perplexing to beginners; its purpose is to allow you to search for strings that contain slashes (/) or, for that matter, any special characters. The general form of the command is:-
FIND delimiter string delimiter [lineno, lineno] [,A] In the general form, symbols within a pair of brackets are optional qualifiers of the command.

REP Command:
This command replaces a specified string in your source program with a different specified string.
REP/OLD/NEW RETURN:
Replaces the first occurance of the string OLD with the string NEW.
REP/OLD/NEW/xx,yy RETURN:
Replaces the first occurance of the string OLD between statements number xx to yy with the string NEW.
REP/OLD/NEW/,A RETURN:
Replaces all the occurances of the string OLD with the string NEW.
REP/OLD/NEW/xx,yy,Q RETURN:
Displays, in turn, each occurance of the string OLD between statements xx and yy. Q stands for "query." To replace the displayed OLD with NEW, type Y, then RETURN. To retain the displayed OLD, press RETURN.
In these examples, the string OLD AND NEW are delimited by the character "/". As with the FIND command, any character except space, tab and RETURN, can be used as the delimiter.

ATARI ASSEMBLER EDITOR CARTRIDGE

For example, the command REP+RTS +BRK+,A replaces all occurrences of RTS with BRK. The delimiter is the character "+".

The general form of this command is:- [Q] REP delimiter OLD delimiter NEW delimiter [lineno,lineno] [,A]

In general form, symbols within a pair of brackets are optional qualifiers of the command and the symbols within braces (A and Q) are alternatives.

COMMANDS TO SAVE AND RETRIEVE PROGRAMS LIST:

Saves or displays a source program.

PRINT is the same as LIST, but omits line numbers.

ENTER: retrieves a source program.

SAVE: saves an object program.

LOAD: retrives an object program.

With each of these commands there is a parameter that specfies the device that is the source or destination of the program that is to be saved, displayed or retrieved. The possible devices are different for different commands, and the default device is also different. Some of the commands have optional parameters that limit the application of the command to specified parts of the program. The parameter that specifies the device that is the source or destination of the program is written as follows:-

#E: is the screen editor.

#P: is the printer.

#C: is the Program Recorder.

#D[n]:FILENAME is a disk drive. n is 1,2,3,4, or 8 (RAMdisk).

#D:FILENAME is interpreted as D1: A program save on or retrived from a diskette must be named ie:- (FILENAME)

LIST Command. Example:-

LIST#D:MYFILE (diskdrive).

LIST#C:(Program Recorder).

LIST#P:(Printer).

LIST#E:(Editor/TV Screen)

These commands are used to save a source file.

You can of course LIST some line numbers as with BASIC ie:-

LIST#D:MYFILE,100,500 would list lines 100 to 500 to disk from a source code which starts at line 10 till 3000.

ENTER Command:

Used to enter a source file Example:-

ENTER#D:MYFILE ENTER#C:

Only a fool would try to ENTER a source program from the EDITOR, PRINTER.

PRINT Command:

This is the same as LIST, except that it prints statements without statement numbers. (line numbers)

SAVE Command:

FORMAT Example:

S A V E # C : < 1 2 3 5 , 1 7 3 6

ATARI ASSEMBLER EDITOR CARTRIDGE

SAVE#D:MYFILE<123,1736
Saves an object program residing in hex address 1 to address 2 on cassette or disk, the commands are:-
SAVE#D:<address1,address2
LOAD Command FORMAT: Command
LOAD#C: LOAD#D:MYFILE:
These commands will reload the memory locations address1 to address2 with the contents that were previously saved. The numbers address1 and address2 are those that were given in the original SAVE command.
USING THE ASSEMBLER
ASM Command:
This Chapeter is long and covers lots of information. I shall brush over some commands.
ASM RETURN:
On receiving this command, the Assembler translates the source program in the edit text buffer into object code and writes the object code into the memory locations specified in the source program. When this process is completed, the assembled program is displayed on screen. You may specify that the assembled program is to be stored directly on diskette, useing any name (subject to the restrictions of DOS).
In the general form of the ASM command.
Example:-
ASM#D:SOURCE,P:,D2:SEMBLED.OBJ RETURN:
The above command takes the source program that you had previously stored on disk and called SOURCE, assembles it, list the assembled form on the printer, and recordes on the diskette the machine code translation of the program (the object program). The object program is given the name "SEMBLED.OBJ".
Note that the commands of this form store the machine code on disk, not in computer RAM.
The default value is the screen (#E:). The other possibilities are the printer (#P:), the program recorder (#C:), and the disk drive (#D[n]:NAME.
To make a default selection, enter a comma, as in the following usefull command:-
ASM,#P: RETURN:
The above command takes the source program from the default edit text buffer, assembles and lists it on the printer as before, and stores the machine code (object program) directly into computer RAM.
DIRECTIVES operations:
Summary Directives are instructions to the Assembler. Directives do not, in general, produce any assembled code, but they affect the way the Assembler

ATARI ASSEMBLER EDITOR CARTRIDGE

assembles other instructions during the assembly process. Directives are also called psudo operations or pseudo ops. You will have to read the Assembler manual for more information on Directives.

DEBUGGING:

The Debugger allows you to follow the operation of an object program in detail and make minor changes in it. A knowledge of machine code is helpful when you use the debugger, but it is not essential. The debugger is able to convert machine code into assembly language (disassemble), so you can make code alterations at particular memory locations. All numbers used by the Debugger, both input and output, are hexadecimal. The Debugger is called from the editor by typing:-

BUG RETURN:

This produces on screen:- DEBUG:

The command to return to the Writer/Editor is:- X RETURN.

DEBUG COMMANDS:

DR Display Registers

CR Changes Registers

D or Dmmmm Display Memory (mmmm = Memory Range)

C or Cmmmm Changes Memory (Ditto)

Mmmmm Move Memory

Vmmmm Verify Memory

L or Lmmmm List Memory with Disassembly.

A: Assemble One Instruction Into Memory

Tmmmm: Trace Operation

S or Smmmm: Single-Step Operation

Gmmmm: Go (Execute Program)

X: Return to EDITOR

BREAK:

Pressing the BREAK key halts operations.

EXAMPLES: DR Display Registers

DR RETURN A=BA X=12 Y=34 P=BO S=DF

CHANGE REGISTERS CR<1,2,3,4,5 RETURN

THE EFFECT OF THE COMMAND ABOVE IS TO SET THE CONTENTS OF REGISTERS A,X,Y,P, and S to 1,2,3,4, and 5.

You can skip registers by using commas after the <.

For example: CR<,,,,E2 RETURN

D or DISPLAY MEMORY

Dmmmm,yyyy where yyyy is less than or equal to mmm shows the contents of address mmmm Examples:-

D5000,0 = 5000 A9

D5000 = A9 03 18 E5 F0 4C 23 91

D RETURN 5008 18 41 54 41 52 49 20 20

D5000,500F RETURN

5000 A9 03 18 E5 F0 4C 23 91

500B 18 41 54 41 52 49 20 20

ATARI ASSEMBLER EDITOR CARTRIDGE

C or Cmmmm Change Memory
Cmmmm <yy changes the contents of address mmmm to yy
Examples:- C5001<23 RETURN
C500B<21,EF RETURN
C700B<31,,,87 RETURN
Mmmmm Move Memory
Mmmmm<yyyy,zzzz copys memory from yyyy to memory starting at mmmm.
Address mmmm must be less than yyyy or greater than zzzz
Example:- M1230<5000,500F RETURN
Vmmmm Verify Memory
Vmmmm<yyyy,zzzz compares memory yyyy to zzzz with memory starting at mmmm, and shows mismatches.
Example:-V7000<7100,7123 RETURN
L or Lmmmm List Memory with Disassembly
This command allows you to look at any block of memory in disassembled form.
Examples:- L7000 RETURN List a screen page (20 lines of code) starting at memory location 7000.
Pressing the BREAK key during listing halts the listing.
L RETURN
This form of command list a screen page starting at the instruction last shown, plus 1.
L7000,0 RETURN
These forms list the instructions at address.
L7000,7000 RETURN 7000 only
L7000,6000 RETURN
L345,567 RETURN This form list address 345 through 567
Note The command Lmmmm differs from Dmmmm in that Lmmmm disassembles the contents of memory.
Example:-
L5000,0 RETURN
5000 A9 03 LDA #$03
This example shows that the Debugger examined the contents of memory address 5000 and disassembled A9 to LDA. Since A9 must have a one-byte operand, the Debugger made the next byte the operand. Therefore, although the Debugger was "asked" for the content of location 5000, it shows a certain amount of intelligence and replied by showing the instruction that started at address 5000. To illustrate this, the number 03 corresponds to no machine code instruction, so the Debugger would interpret 03 as an illegal instruction. However, if the first instruction you wrote was LDA $8A, then you would have obtained the following, apparently inconsistent, results while debugging. Example:-L5000,00 A9 8A LDA #$8A.
Because the disassembler starts dis-

ATARI ASSEMBLER EDITOR CARTRIDGE

assembling from the first address you specify, you have to take care that the first address contains the first byte of a "real" instruction.

DEBUG

To assemble an instruction, first enter the address at which you wish to have the machine code inserted. The number that you enter will be interpreted as a hex address. Now type < followed by at least one space, then the instruction.

A RETURN 5001<LDA $1234 RETURN

5001 AC3412 Computer Responds.

<INY RETURN

5004 CB Computer Responds.

Since the mini-assembler assembles only one instruction at a time, it cannot refer to another instruction. Therefore, it cannot interpret a label. Consequently, labels are not legal in the mini-assembler.

You can use the Directives:- BYTE, DBYTE, and WORD.

Gmmmm Go (Execute Program)

This command executes instructions starting at mmmm.For example:-

G7B00 RETURN

Executes instructions starting at location 7B00.

Execution continues indefinitely. Execution is stoped by pressing the BREAK key.

Tmmmm Trace Operation

This command has the same effect as Gmmmm, except that after execution of each instruction the screen shows the instruction address, the instruction in machine code, the instruction in assembly language (disassembled by the debugger-not necessarily the same as you wrote it in assembly language) and the values of Registers A,X,Y,P and S.

The execution stops at a BRK instruction (machine code 00) or when you press the BREAK key on the keyboard.

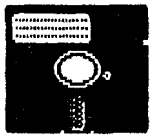
S or Smmmm Step Operation

This command has the same effect as T or Tmmmm, except that only one instruction is executed. To step through a program, type:-S RETURN repeatedly after the first command of Smmmm RETURN

X EXIT To return to the Editor type:-X RETURN

Compiled From the Atari Assembler Editor Users Manual.

M. Tomlin. June 1996 Textpro Ver. 1.2a (Document is 11 Pages to print out).



DISK CONTENT

This issue disk content was supplied by M. Tomlin (WACO), and by A. Thompson (TOMO)

This issue disk is full of programs and I am sure come in handy.

Most of these programs have been supplied by Mr. Tomlin and they include:

EPSONBAS.GEN---for the Citizen ABC colour printer

P6WRITER.COM,

TINITEXT.BAS

VIEWDOC.BAS

FORMAT.BAS

LOTTERY.BAS

CIRCUIT.BAS----that's on Side B

You will also find the doc files for these basic programs on the disk.

TWAUG has also included an Art program called PIXEL ARTIST.

Andrew Thompson has supplied us with his Coding Caper programs:

MUSICAL REFERENCE and MULTIPLE FONT OUTPUT for the Atari 1029

All in all a good selection of programs.

ENJOY

ADVERTISING USER GROUPS

LACE

The LONDON ATARI COMPUTER ENTHUSIASTS

As a member of LACE you will receive a monthly newsletter and have access to a monthly meeting. They also support the ST and keep a large selection of ST and 8-bit PD software.

The membership fee is £8.00 annually

for more information contact:

Mr. Roger Lacey
LACE Secretary
41 Henryson Road
Crofton Park
London SE4 1HL
Tel.: 0181 - 690 2548

O.H.A.U.G.

The OL'HACKERS ATARI USER GROUP INC.

O.H.A.U.G. is an all 8-bit user group in the STATE of NEW YORK.

They are producing a bi-monthly double sided disk based newsletter. The disk comes with its own printing utility, which lets you read the content of the disk, one screen page at the time, and/or you can make a hard copy of the disk, in one, two or three columns and 6 to 8 lines to the inch. A large PD Library is available.

Contact:
Mr.Ron Fetzer
O.H.A.U.G.
Secretary & Treasurer
22 Monaco Avenue
Elmont, N.Y. 11003
USA