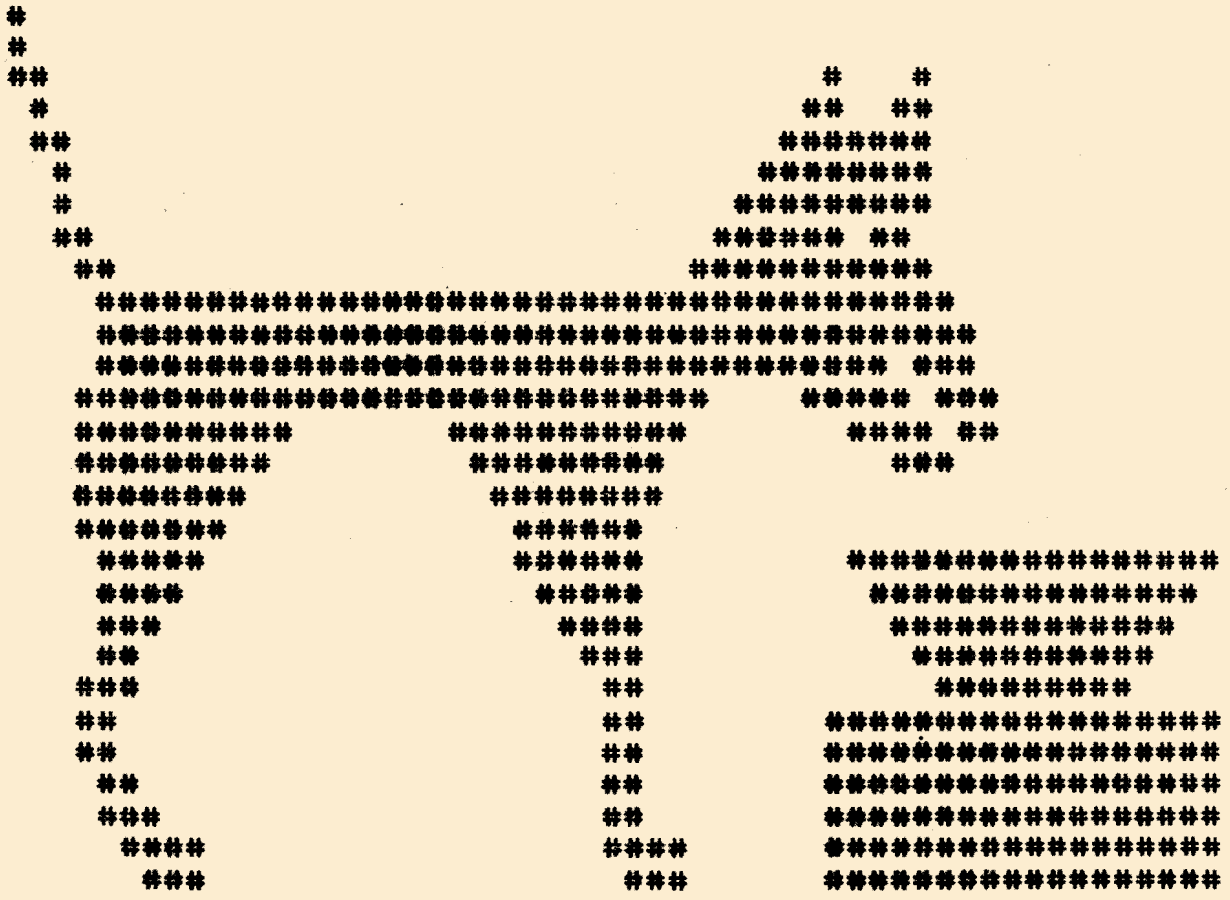
REGISTERED BY AUSTRALIA POST -
Publication No. NBG4250.

DREAMER No.15 NOV, '81.

N.S.W. 6800 USERS GROUP,
[redacted]



BIG DOG

CLEARANCE

DREAM INVADERS: There are still a few copies of this popular, fast moving action game, based on Space Invaders. Sorry, no cassetes left. Instructions (incl. 2K RAM expansion notes) plus HEX listing...$5.00. Fully commented 6800 Assembly listing (18 pages) .....$5.00.

CHIPOS MANUAL: Due to continuing demand, a new print run has been made, but increased printing costs have forced the price up a bit to $6.00. (Foreign orders, please add $1.00) The manual contains a fully commented source listing of the CHIPOS EPROM, plus valuable data for the machine-code programmer or student thereof, etc.

Send to: 'DREAMWARE', PO Box 343, BELMONT, VIC., 3216.

HARDWARE FROM DREAMSOFT !!

DREAMSOFT have developed a new DREAM Expansion Board.

FEATURES :-

* Full 8K RAM - no need to sacrifice RAM space to accommodate EPROMS.
* Sockets for 3 x 2716 EPROMS - one at 1800-1FFF to take our No.1 Package. That's 6K of EPROM Space!
* 2 PIA's.
* Data and Address lines are fully buffered and can drive even more devices so any expansion you may already have will not be obselete.
* Same size as DREAM board and can be mounted above or below it.
* Single +5v Supply - fully equipped board draws less than 2 amps.
* Double sided, Plated-thru holes - no links to install.
* Can be equipped to suit your needs and your budget.
* Will retail for $25 incl. P&P. (board only).

Occupies the entire bottom 16K of address space (0000-3FFF).
Full assembly instructions are supplied with each PCB.

We expect that this will be written up as a project in the Feb. 1982 issue of Electronics Australia.

* NB We do not supply components

The board will be available in December 1981. Place your order now to avoid delays.

Mail this coupon now

- -

To DREAMSOFT
P.O. BOX 139,
MITCHAM VIC. 3132

Please RUSH . . . . . DREAMSOFT Expansion Boards at $25 each.

A Cheque/Money Order is enclosed for $

SENT TO: Name _____

Address _____

_____Postcode_____

No. 15 NOVEMBER, 1981.

Well, it finally happened! Garry has run out of things to say, so this space will have to remain blank for this month. (Well, almost.)

Don't forget to send in those programs and articles that you have been working on.

Next month we will have Part 3 of Lindsay Ford's Assembly Language Articles, a program to turn your DREAM into a FLOATING POINT CALCULATOR, a modified Joystick Service routine, and lots more new games.

HAPPY DREAMING,
GARRY and GRAEME.

****

ADVERTISING

FOR SALE: GARRY'S PRINTER. It is a Centronics 779, the same as the one we print the programs on for the DREAMER. (Yes, we had two of them.) These were sold by Tandy a couple of years back for $1799. It can be yours for $500.00, (or a close offer), complete with Software and instructions on how to hook it up to your DREAM. PHONE GARRY on [redacted] or GRAEME on [redacted] (Or write to us C/- [redacted]

++++++++++++++++

FOR SALE: DREAM 6800 with 3K RAM, 3A Power Supply, Joystick, 2K EPROM, Bags of software, Works well, looks good. (Selling because I now have a big system.) Price negotiable. Phone RAY SCHMIDT, [redacted] or [redacted]

****

ASSEMBLY LANGUAGE AND THE ROLLING CHECKSUM

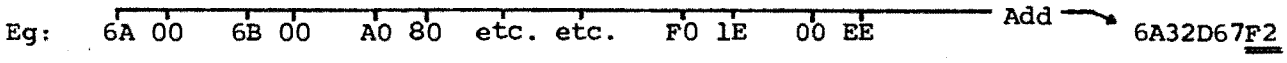
PART 2

By Lindsay R. Ford,
"Dreamcards",
8 Highland Crt.,
Eltham North 3095 Vic.

In the last article in this series we went into some detail about the basics of Assembly Language as a preliminary to using it to build up an actual programme. If you haven't read that article (or if you've forgotten what it explained) then read it before you go on. The author takes no responsibility for brain damage caused to people who fail to heed this warning!!!

Imagine that you've just got a job as Chief Programme Designer with Dream Porn Inc. (purveyors of Chip 8 depravity to the rich and famous) and you've been asked to solve their most ticklish problem. It seems that some customers have been caught in embarrass ing positions when their 2K programme "The Auto-Seductor" went haywire half way through. Now you can't expect a rich and famous customer to check every byte of a 2K programme each time he loads it from tape, so your task is to devise a routine that will automatically check for errors before the programme is run.

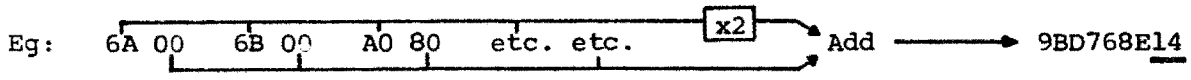
A simple way you could approach this is to add each byte in the programme on the assumption that the total would always be the same if the load was correct. It would soon become obvious, though, that the total would be a number of rather sizeable proportions. A compromise would therefore be to concentrate only on the last 2 digits;



Now if an error occurred in the loading of the programme (such as if noise in the cassette deck turned the 'A080' into 'B080') then the last two digits of the total would change, indicating the fault. The only chance it would have of escaping detection would be if the total changed by a multiple of 100 (Hex.), a situation that would require odds in its favour of 1 in 256 and at least two errors.

This level of accuracy is quite acceptable and many programmes contain just such a routine to verify the software. It is called a simple checksum, but such routines have a shortcoming with Chip 8 in that they will not detect byte transposition errors. As an instance of this let's imagine that the 'A080' instruction in our example got 'transposed' into '80A0'. The result of this change to the programme would be disastrous, but the error would not be detected by a simple checksum.

The best way to pick up a byte transposition error is to make the checksum differentiate between the first and second bytes of each Chip 8 instruction, doubling the first byte before it's added to the checksum total.



I have called the routine to do this a rolling checksum (for want of a better name) and you can work out for yourself how the total changes if an error occurs. It's disadvantage is that a single error can now cause a change of 100 (Hex.) in the total and thereby escape detection, although the odds are still 256 to 1 against it.

DESIGNING THE ROUTINE;

Having worked out what we want the routine to do, let's try and put it together a step at a time. We'll start it at $0080 for this example, but we'll aim for a routine that can be located anywhere in memory without modification as this would allow us to put it on the end of any Chip 8 programme without the bother of modifying it to suit the new location (such routines are said to be 'fully relocatable').

1) Index and Clear:

To begin with we ought to define our starting parameters. Let's say we want the Checksum to add every byte between $0200 and $07FF (2K of programme - there's not much point in

starting below $0200 as we'd be into the video display buffer memory and this usually only contains random junk after a load is done). We'll also need 8 bits of memory in which to store our 2 digit total (Accumulator B will do) and this will have to be set to #00 before we start adding things to it.

```
So,     0080     CE 0200          LDX $#0200
           3     5F               CLR B
```

Now we've already discussed the LDX instruction in the last part of this article - in this case we've used it to point the MPU to the first byte of programme we want added to the Checksum. The second instruction (CLR B) clears Accumulator B, setting its 8 bits of memory to zero. Needless to say, this is an instruction in the 'inherent' address set and so it requires no operand.

2) Add and Double:

Now our intention was to add the doubled value of the first byte of Chip 8 to the Checksum total in Accumulator B, then to add the second byte to the result. The best way to effectively double the value of the first byte is simply to add it twice over,

```
So,     0084     EB 00            ADD B  $X
           6     EB 00            ADD B  $X
           8     EB 01            ADD B  $X+1
```

e instruction described by the mnemonic ADD B selects the byte of data at the address rmed by adding the number in the Index Register ($X) and the number in the operand remember our discussion about 'indexed addressing'?) and adds it to the contents of Accumulator B. Thus, as the number in the Index Register at this stage is #0200, the 'EB 00' instruction will add the byte at $0200 (#0200 + 00) and the 'EB 01' instruction will add the byte at $0201 (#0200 + 01). Note that the actual contents of the Index Register are not affected by this operation.

Let's suppose that address $0200-01 contains the Chip 8 instruction 'A080' - the ADD B instructions set out above will therefore perform the calculation 'A0 + A0 + 80 = 1C0'. As Accumulator B can only hold 8 bits it will only retain the lowest two digits of this sum and, when the instructions have been performed, will contain #C0.

(Note: Some readers will be aware that a two digit number stored in an Accumulator can be doubled in value using a Shift Left instruction, rather than by double addition. This has not been done in our example as this would double the actual total rather than just the first byte of each Chip 8 instruction).

3) Increment Index

Once we've finished with addition of the first Chip 8 instruction we want the routine o repeat the same function on the second instruction. Now we know that the Index gister contains #0200 at this stage, so all we need to do is to increment this by 2 to make it point to the second Chip 8 instruction at $0202. This is done with the 'Increment Index Register' instruction;

```
         008A     08               INX
            B     08               INX
```

This instruction is inherently indexed (like the CLR B instruction described above) so it requires no operand. Each time it is encountered by the programme it adds #01 to the contents of the Index Register, in this case increasing it by #02 as we have used it twice.

4) Check Index

At this point in the routine we've checked one Chip 8 instruction and the Index Register now points to the next one. We could therefore keep the whole procedure repeating itself by telling the programme to return to the ADD B instruction at $0084, but this would cause an endless loop with the computer happily adding away until you got sick of it and hit the reset button. What we want, therefore, is an instruction that will force the programme to return to $0084 only as long as it is still adding relevant material. This is achieved with;

```
         008C     8C 0800          CPX $#0800
```

This 'Compare Index Register' instruction tests the contents of the Index Register by subtracting the value of the operand from it (although the Index Register and its contents are unaffected by this operation). If the result of this test is zero (ie: the Index Register had worked its way up to $0800, the location at which we want our checksum to end) then a flag in the MPU known as the Zero Status Flag is set to #1. If the result is other than zero the flag remains in its usual state, set to #0.

5) Branch

Now it's all very well turning flags on and off, but this doesn't help us control the programme flow unless we follow the CPX instruction with something else that detects and acts on the state of the flag. The particular instruction we use to achieve this is the 'Branch if not Equal' (to zero) instruction;

```
008F        26 F3                    BNE $0084
```

What happens when this instruction is encountered is that the computer looks at the Zero Status Flag and if it finds that the flag is set to #1 (ie: our previous CPX instruction has detected that the Index Register contains #0800) it causes the programme to jump over the operand to the instruction at (in this case) $0091. Where the BNE instruction finds that the flag is set to #0 (ie: Index Register not at #0800) it functions differently, causing the programme to return or advance to a location specified by the operand. Now this brings us to the question of how far backwards or forwards the programme will go and this is of sufficient importance to consider separately (particularly considering that 6800/6802 Assembly Language contains 16 different Branch instructions!)

Calculating Branch Offset;

The two digit operand following a Branch instruction is not in itself a memory address, but relates to the number of bytes of programme separating the Branch instruction and the location to be branched to. As these instructions are therefore independant of their actual position in the computers memory they are extremely useful in designing relocatable programmes.

If you look at a few programming textbooks you will see that various formulae are given to calculate what the operand should be in a given case (the branch offset), but probably the most foolproof way to work it out is to count the bytes involved. The way you do this is by calling the location occupied by the operand of the Branch instruction 'FF' (it doesn't matter what its actual address is) and then, to branch backwards, counting down from 'FF' (in hexadecimal) for each byte of programme until you reach the location you want to get to. To branch forward you adopt the same procedure, but count up from 'FF' rather than down.

In our routine we want to get to $0084 so that the programme repeats the additions, hence a backwards branch is required. By counting it out we get a branch offset of 'F3' and so this is the operand (don't just accept this - count it out yourself).

If you give this a bit of thought you will realise that Branch instructions must be limited in the distance they can go. In fact the limit is 128 bytes of programme material (80 Hex.) and thus if we used '80' as an operand the programme would branch back that far, whereas '7F' would cause it to branch forward the same distance.

If you're going to learn Assembly Language it is important for you to be able to work out branch offset and so you should try out a few different examples. Once you're proficient at it you can take the lazy way out and use the Branch Offset Calculator routines in Michael Bauer's "Chip 8" book or in the "Dreamsoft No.1 Eprom", but not before you know how to do it yourself.

6) Display the Result;

Wham!! The Index Register has finally made its way up to $0800 and Accumulator B now contains the Checksum total, but what the hell do you do with it?

Well, if you're intending to use the Checksum independant of the main programme (ie: by keying in 0080 FN 3 to run it) then the answer is simple. There are a number of subroutines in your Chip 8 interpreter Eprom that allow you to display a two digit number, so all you do is use 'Jump to Subroutine' instructions (similar to Chip 8 '2XXX' "Goto Subroutine" instructions except that they can address 64K of memory) to get to them. Now some of these subroutines require various parameters to be set up

before they will work, but these are beyond the scope of this article. I will set out the basic routine you will need, but for full details on each one Michael Bauer's "Chip 8" book is a must.

```
0091    CE 01C8     LDX $#01C8   ┐
   4    86 FF       LDA A  #FF   ┘ White out video display window
   6    BD C07D     JSR "FILL"   ┘
   9    86 1C       LDA A  #1C   ┐
   B    BD C3E0     JSR "CURS. 1"┘ Set Horiz. Display Co-ordinate
   E    17          TBA
   F    BD C3CA     JSR "SHOBYTE"  Display 2 digit Number in Acc. A.
  A2    20 FE       BRA $00A2      Loop ad infinitum.
```

Well, that wasn't too hard, was it? After not terribly much brain-strain we've got ourselves a reasonably complicated Assembly Language routine worked out. For the next month or so you can play with it (0080 FN 3 to run) by making changes to the data in the programme area ($0200-07FF) and see how this affects the Checksum total. Another little task I'll set for you is (if you've already got Michael Bauer's book) to work out why I started the routine off putting the checksum total in Accumulator B when to finally display it you've got to have it in Accumulator A. Next month I'll answer this one for you, explain the various instructions used above and show you how to make the whole thing fully automatic.

****

PROGRAM HINTS

E. WITTE, [redacted]

Instead of method A shown below, use method B and save two bytes of space.

METHOD A

```
0300  3A01
0302  1308
0304  7BFF
0306  130A
0308  7B01
030A  DAB5
030C  ....
```

METHOD B

```
0300  3A01
0302  1306
0304  7BFE
0306  7B01
0308  DAB5
030A  ....
```

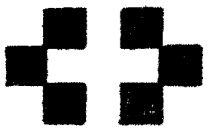
You can save time in program loops by useing only one instruction to display a character and erase the same character. This method is very useful when showing an object move across the screen. e.g. as in UFO INTERCEPT. Shown below is the method:

```
0200  6A00
0202  6B00
0204  A220
0206  DAB3
0208  A230
020A  DAB3
020C  7A01
020E  6C03
0210  FC18
0212  120A

0220  60F0
0222  6000

0230  5088
0232  5000
```


= 60F060


= 508850


= 508850 overlayed on 60F060


= After DISPLAY inst. (020A)

****

BIG DOG. (0080 - 0400)

ROGER GRAHAM,
[redacted]

Here is a computer-animation program that readers should enjoy, called "Big Dog". I have noticed that programs so far offered have not included much of this animation material at all.... just a few small bits like 'Absent-Minded Dog' in the May issue. So try my "Big Dog" for some variety, his lick is worse than his BYTE!

In brief, the program draws a big dog on the screen. He licks up food from his dish, swallows, gets fatter, wags his tail, and blinks his eyes.

Some addresses of interest that you can play around with to get different effects, are:-

0349 (How many licks before swallowing.)
026F (How fat dog gets before he stops eating.)
02E9 (How many times his tail wags.)
0371 (Speed of tongue lap.)
037D (Speed of tail wag.)
02FF (How many times he blinks.)
0381 (Time between blinks.)

In addition, you can change his bowl from a plain one to one labelled 'PAL', or 'MILK', by making the following amendments.

To make doggie eat PAL

Change	03F8	to	84B5
	03FA		84BD
	0086		2FAF
	0088		2FA3

To make doggie drink MILK

Change	03E6	to	0302
	03E8		0202
	03EA		0202
	03F6		FF4A
	03F8		AAAA

Change	03FA	to	EAEA
	0084		FFED
	0086		EBE7
	0088		EB2D

```
0200   6000 A384 D00E 6108   620B A392 D123 7108
0210   D123 7108 D123 7108   A395 6304 D13A 7108
0220   A39F D13A 620E A3A9   D020 6108 A3B9 D120
0230   7108 A3C9 D124 7108   A3CD D120 7108 A3DD
0240   D120 7108 A3ED D120   7108 6213 A080 D12B
0250   2306 650F 6708 6810   6601 6401 A08B F61E
0260   D751 F41E D851 2380   2306 7602 7501 3516
0270   125C 6E00 A384 D003   A09C D003 237C D005
0280   A0A1 D005 237C D007   A0A8 D007 237C D00A
0290   A0B2 D00A 237C D00B   A0BD D00B 237C D007
02A0   A0C8 D007 237C D005   A0CF D005 237C D005
02B0   A0C8 D005 237C D007   A0BD D007 237C D00B
02C0   A0B2 D00B 237C D00A   A0A8 D00A 237C D007
02D0   A0A1 D007 237C D005   A09C D005 237C D003
02E0   A384 D003 237C 7E01   3E08 1276 6400 6D27
02F0   6009 A09A DD01 2370   DD01 7401 2380 3405

0300   12F4 00E0 1200 6B00   692C 6A10 A08B D9A1
0310   2370 6A11 D9A1 2370   6A12 D9A1 2370 692D
0320   D9A1 2370 692E D9A1   2370 D9A1 2370 692D
0330   D9A1 2370 692C D9A1   6A11 2370 D9A1 6A10
0340   2370 D9A1 2370 7B01   3B08 130E 6126 620E
0350   D121 2370 D121 6123   D121 2370 D121 6122
0360   D121 2370 D121 6121   D121 2370 D121 00EE
0370   6302 F315 F307 3300   1374 00EE 6302 1372
0380   6310 1372 0080 C040   6020 3010 1018 0C07
0390   0707 FFFF FF02 060F   1F3F 7EFF FFFF FF20
03A0   60E0 E0E0 C0E0 F0F8   B80F 0F0F 0F0F 0707
03B0   0706 0E0C 0C06 0703   01FF FCF8 F0E0 C080
03C0   0000 0000 0000 00C0   C0FF 0703 01FF FFFE
03D0   FEFC FC7C 3C1C 0C0C   0C0C 0C0F 0787 0100
03E0   0000 0100 0000 0003   0303 0303 03DC EC70
03F0   0000 FFFF 7F3F 1FFF   FFFF FFFF FFEA F290
```

(See Page 7 for 0080 - 0100)

SILENT KEYWAIT INPUT.

FRANK REES,
[redacted]

This routine is for a silent key input to avoid a 'bleep' when it is undesirable. E.G., - in musical programs.

```
0250    0000 DF12 BDC2 87A6   012B 0748 2AF9 6D00
0260    200B BDC2 977D 0018   26EA 9731 DE12 7E02
0270    00** ****
```

The program is relocatable anywhere without change. If you use it as a sub-routine in a program, remember that it is in machine code, and use a 0xxx instruction to call it, and a 39 (RTS) to return to the main program.

Location 026B shows 31. This is the memory location (0031) of Variable 1. This can be changed to any other one byte location, and is the place where the key struck is stored.

The last instruction at 026E is 7E (JUMP TO) 0200. This should be replaced with 39 if used as a subroutine, or 0200 changed to the location you wish the program to jump to next.

****

CREATE YOUR OWN SOUND EFFECTS (0200 - 0300)

R. FULLARTON

This program allows you to test different codes for use in a sound effects routine.

Simply key in a four digit code and it will be displayed on the screen and played. Then key in 0 and a new code, or key in 1 and the previous code will be played again.

```
0200   00E0 6614 6710 F00A   F029 229A FA0A FA29
0210   229A 0238 80A4 8A00   A244 F055 661E F00A
0220   F029 229A FA0A FA29   229A 0238 80A4 8A00
0230   A245 F055 0243 1287   B600 3048 4848 48B7
0240   0030 39CE 9566 BD02   4CC6 0339 BD02 5CDF
0250   2696 27B7 8022 9626   7E02 7100 C63C F780
0260   2139 C604 F780 237F   8022 7F80 23B7 8022
0270   39CE 8020 C638 E701   C6FF E700 C63C E701
0280   A700 C634 E701 39FE   0A4E 0112 3402 9112
0290   00CE E000 BD02 4CC6   0339 D675 7605 00EE
```

****

BIG DOG (Cont) (0080 - 0100)

```
0080   FFFE FCF8 F0FF FFFF   FFFF FF80 03F8 07FC
0090   0FFE 1FFF 1FFE 0FFC   07F8 9000 4040 4040
00A0   6020 2020 2020 2030   1010 1010 1010 1010
00B0   1018 0808 0808 0808   0808 0808 0C04 0404
00C0   0404 0404 0404 0404   0202 0202 0202 0600
00D0   0101 0103 5145 0145   1352 D569 C555 5145
00E0   3155 5155 5555 5555   6B89 D5B1 0101 51D4
00F0   0591 5115 4125 5545   7DF8 7D56 4125 D54A
```

****

ERRATTA

In 'COMPUCROSSES', JULY DREAMER,
The LEFT score is YOURS, the MIDDLE score is the COMPUTER'S, the RIGHT score is the number of DRAWN GAMES played. Thank you Mr. D. Chaves.

****

SHEEPDOG

(0080 - 0400)

K. SEMRAD,
[redacted]

You control the 'sheepdog' with your Joystick. Every time the dog barks, (Key F) the sheep closest to him will move away from him. Your task is to get all the sheep in the little paddock in the top left corner, but watch out, the sheep are a lively mob, except for one, it is lame. They try to get out of reach of the dog, and once one is gone, all you see is a dot in the 'distance'. (You might want to fiddle with location 0257 (hor.) and 0260 (vert.) to avoid ugly overlaps when the dog barks.)

Happy droving!

```
0090   280E 00E0 6A02 6B0A   F729 2296 6B18 A080
00A0   F933 F265 F029 2296   F129 2296 F229 2296
00B0   6E00 6A0A 6B0A 20CC   3E07 10B6 6A16 6B18
00C0   20CC 3E0C 10C0 F00A   00E0 1200 A0DA FE1E
00D0   F065 03CC 2296 7E01   00EE 0100 0304 0201
00E0   0508 0407 0600 E7CE   F3CE 935C F24E B7DE
00F0   D6DD B75A BBDE D7DD   0060 6000 F0F0 F0F0

0200   0204 1268 C63B F780   13C6 61F7 8012 C63F
0210   F780 13CE 0219 DF00   397A 0020 7A00 217D
0220   8012 7C00 1696 1684   0327 013B 86FF 973C
0230   973D B680 1284 40B7   8012 8A21 B780 12C6
0240   4AB6 8012 4646 2403   7C00 3C46 2403 7C00
0250   3D5A 26ED 963C 800C   2C01 4F97 3C96 3D80
0260   0E2C 014F 4797 3D3B   A30B 6001 6100 6209
0270   D011 D021 A285 D101   D201 7001 3009 1274
0280   129C 6200 A080 F21E   F165 A285 D011 7202
0290   3212 1284 00EE DAB5   7A04 00EE 6200 C00F
02A0   C10F 7028 7108 A080   F21E F155 7202 3212
02B0   129E 6700 6800 6900   2282 8AC0 8BD0 A0F8
02C0   DAB4 6E0F EE9E 137C   7901 A0FC DAB4 03D4
02D0   2282 6200 6E04 640A   85A0 86B0 A080 F21E
02E0   F165 3000 403F 1370   3100 411F 1370 8A05
02F0   3F00 1304 03C4 8AE5   3F00 1366 7001 403F

0300   134E 132C 8AE5 3F00   1366 70FF 4000 134E
0310   8045 3F01 131A 8044   132C 8145 3F00 1328
0320   6000 6100 7801 1366   8144 8044 8B15 3F00
0330   1342 03C8 8BE5 3F00   1366 7101 411F 134E
0340   1366 8BE5 3F00 1366   71FF 3100 1352 7701
0350   1366 8145 3F01 135C   8144 1366 8045 3F01
0360   1320 8044 8144 8A50   8B60 A080 F21E F155
0370   7202 3212 12D4 2282   A0FC DAB4 2282 C20E
0380   A080 F21E F165 3000   403F 13AC 3100 411F
0390   13AC 03DD 3000 403F   13A8 3100 411F 13A8
03A0   A080 F21E F155 13AC   7701 13A0 2282 5AC0
03B0   13B6 9BD0 12C2 A0F8   DAB4 8874 4809 1092
03C0   8875 12BA 7000 3A39   7000 3B39 9630 CE00
03D0   E47E C198 C640 8603   9721 7EC2 E5BD C132
03E0   8108 22F9 8100 2604   7A00 3039 8101 2604
03F0   7A00 3139 8102 2604   7C00 3139 7C00 3039
```

****

SHIFT SCREEN

BRETT RITTER,
[redacted]

Here is a machine code program that I am sure some industrious readers will want to incorporate into a program, or even be inspired to write a program around it.

What it does is shift the screen, all of it, one dot left. When it does, it introduces zero's on the right side of the screen and erases the left most column. Therefore, if you run it 64 times continuously, you can sit back and watch the screen roll past and disappear. By eliminating the clear carry, (0C), changed to 01, the whole screen will wrap around and not be lost. For any object to remain stationary, it must be shifted right immediately before or after the screen shift, preferably before. I have also written a program to shift the screen right, which works in the same manner. Both are relocatable without change. If you use them as subroutines, the last instruction must be 3900. If you jump to them from a program, then use 7E (JUMP), xxxx (the address you want to jump to.) If the last instruction is 7E, don't forget to add 00 to create an even number of bytes.

SHIFT SCREEN LEFT

ADDR.	INSTR.	EXPLANATION.
0200	CE 01FF	Load Index Register to point to last screen byte.
0203	C608	Load Accumulator B to 8
0205	0C	Clear, Condition Code Register carry bit
0206	07	Transfer CCR into Accumulator A
0207	06	Transfer Accumulator A into CCR
0208	6900	Rotate left one byte at IR, store carry
020A	07	Transfer CCR into Accumulator A
020B	09	Decrement Index Register
020C	5A	Decrement Accumulator B
020D	C100	Compare B with 00, B-00 = X
020F	26F6	Branch back to 0207 if B-00 ≠ 00
0211	8C 00FF	Compare IR to see if at first byte of screen
0214	26ED	Branch to 0203 if not done
0216	39	Return to main program.

SHIFT SCREEN RIGHT

0200	CE 0100	Load IR to start of screen
0203	C600	Load byte counter to 00
0205	0C	Clear carry
0206	07	CCR to Acc. A
0207	06	Acc. A to CCR
0208	6600	Rotate right
020A	07	CCR to Acc. A
020B	08	Increment IR
020C	5C	Increment byte counter
020D	C108	Check byte counter and
020F	26F6	branch if NOT at edge of screen
0211	8C 0200	Check to see if at end of screen
0214	26ED	Branch if not, continue if at end
0216	39	Return

****

CRAPS

(0080 - 0400)

R. G. DAVIS,
[redacted]

This is the well known game of 'craps'. The rules are very simple. You have two dice, which are 'thrown' by pressing any key.

If you throw a 7 or 11 you WIN on the first throw.

If you throw a 2, 3 or 12 you LOSE on the first throw.

Any other total is saved and becomes your 'point'.

Then, the dice are thrown again. If you throw a 7, you lose.

If you get a total equal to your 'point', you win. THE DICE ARE THROWN UNTIL YOU WIN OR LOSE.

```
0080   00E0 6800 6900 00EE   6800 7908 00EE A0FD
0090   F333 F265 6463 8435   3F00 10A0 F029 D895
00A0   7804 6409 8435 3F00   10AE F129 D895 7804
00B0   F229 D895 00EE 20C2   83E0 20C2 83E4 208E
00C0   00EE CE07 4E07 10C2   4E00 10C2 00EE 6632
00D0   F615 F607 3600 10D2   00EE 83A0 208E 00EE
00E0   83B0 208E 00EE 2088   227E 1264 2088 2288
00F0   1274 D895 7808 00EE   FF0A 1200 CD00 0100

0200   6A78 6B0A 2080 22A8   22BC 20DA 2088 22B2
0210   22BC 20E0 FF0A 2080   20B6 20CE 4302 10EC
0220   4303 10EC 4304 1248   4305 1248 4306 1248
0230   4307 10E6 4308 1248   4309 1248 430A 1248
0240   430B 10E6 430C 10EC   8530 6830 20B6 20CE
0250   9530 125E 4307 126E   6830 208E 124A 2088
0260   227E 2292 20CE 8AB4   4AFA 12D4 12C6 2088
0270   2288 2292 20CE 8AB5   4A00 12D0 12CC A316
0280   20F2 A31B 20F2 00EE   A320 20F2 A325 20F2
0290   00EE 7812 A32A 20F2   A32F 20F2 A334 D895
02A0   7806 A339 D895 00EE   A33E 20F2 A343 20F2
02B0   00EE A348 20F2 A34D   20F2 00EE 780C A352
02C0   D897 7804 00EE 02ED   20CE 1204 02F3 12C8
02D0   02F9 10F8 02FF 10F8   00B6 03FF B701 A77A
02E0   02D8 2708 7C02 DB7C   02DE 20ED 39B6 0359
02F0   7E03 02B6 035A 7E03   02B6 035B 7E03 02B6

0300   035C B702 DBB6 0314   B702 DEB6 0315 B702
0310   D87E 02D9 8028 5555   5555 7D70 5050 5050
0320   8E8A 8A8A EEEE 88EE   28EE EEAA AAAA EA0E
0330   0A0E 0808 E8A8 A8A8   E8EE A4A4 A4A4 CEAA
0340   CEAA CAEA AAAC AAAA   CEA8 CEA8 CEE0 4040
0350   4040 40E0 80E0 20E0   4060 D888 9012 021F
0360   00E8 EAEE 0ABB 9800   00A8 AA8A 0A92 A000
0370   00E8 EEEE 0A92 9000   0088 A48C 0A92 8800
0380   008E A4EA 053A B000   0575 0777 0677 5700
0390   0555 0554 0555 5400   0755 0777 0775 6700
03A0   0255 0564 0565 5400   0277 0557 0657 5700
03B0   2BA8 EEE3 8ABB BBB8   2AA8 AA82 8A92 AA28
03C0   3AA8 EEE3 8A92 ABB8   12A8 AC82 8A92 AA30
03D0   13B8 AAE2 853A ABA8   00AE A6E0 ABB9 8000
03E0   00AA A880 A92A 0000   00EA A4E0 A929 0000
03F0   00AA A280 A928 8000   00AE ECE0 53AB 2000
```

****

TWO'S COMPLEMENT. (0200 - 0400)

J. MARCHINGTON,
[redacted]

This particular program, the second of two involving hexadecimal to binary conversion, should help all who find the concept of two's complement for the binary representation of negative numbers difficult to understand.

Firstly, key in the positive decimal number. This may be any three figure number between 000 and 127 inclusive. (A 'block' has been built into the program so that any attempt to key in numbers larger than 127 will result in unresponsive keypresses.) Note that a normal single figure number must be preceded by two noughts and a two figure number by one nought.

After the third figure has been keyed in, the hexadecimal pair equivalent will be displayed to the right, and to the right again, the corresponding binary number.

Following the appearance of the TWO'S COMPLEMENT block in the centre of the screen, the two's complement of the previously displayed binary number will appear; that is, the noughts will have been replaced by ones, the ones by noughts, and one will have been added to the result. This can be verified by examination. Computation is completed by the appearance of the corresponding hex.digits and, finally, the negative decimal number.

NOTE: The most negative number which can be represented by eight bits is actually -128 (binary 10000000), but it has no positive equivalent since its two's complement is itself.

As with the first program, there is no need to return to the monitor to restart the program. Simply key in new 0 or 1. As before, run C000, FN, 3.

```
0200   6A04 6B00 F10A 00E0  6CFE 8C14 4F00 1212
0210   1204 7C02 2310 4C01  122C F20A 6CF6 8C24
0220   4F00 1226 121A 7C0A  2310 1254 F20A 6CFD
0230   8C24 4F00 1238 122C  7C03 2310 3C02 1254
0240   F30A 6CF8 8C34 4F00  124C 1240 7C08 FC29
0250   DAB5 1266 F30A 6CF6  8C34 4F00 1260 1254
0260   7C0A FC29 DAB5 8710  0318 0318 8170 0318
0270   0318 0318 8174 0318  8174 8720 0318 8270
0280   0318 0318 8274 8214  8324 65F0 8532 0320
0290   660F 8632 232C 233C  7A08 2346 232C 233C
02A0   7A08 2352 2330 2340  6A09 6B08 600D A396
02B0   238E 238E 238E 238E  238E DABD 232C 233C
02C0   6A20 7B10 6DFF 8D35  7D01 83D0 65F0 8532
02D0   0320 8D50 660F 8632  2352 85D0 232C 233C
02E0   6A14 2346 232C 233C  6DFF 8D35 7D01 A3E5
02F0   FD33 F265 6A04 F029  DAB5 7A04 F129 DAB5

0300   7A04 F229 DAB5 6A00  6B1A A3E4 DAB1 1200
0310   FC29 DAB5 7A04 00EE  B600 3748 B700 3739
0320   B600 3544 4444 44B7  0035 3900 6E32 1332
0330   6E4B FE15 FE07 3E00  1334 00EE 6E19 1342
0340   6E32 FE18 00EE F529  DAB5 7A04 F629 DAB5
0350   00EE 670F 8752 6808  237A 6707 8752 6804
0360   237A 6703 8752 6802  237A 6701 8752 6801
0370   237A 8560 4A30 1352  00EE 8785 4F01 1384
0380   6900 1386 6901 F929  DAB5 7A04 00EE DABD
0390   7A08 F01E 00EE FFFF  FFFF FFFF FF88 BABA
03A0   BA88 FFFF F8FD FDFD  FDFF 82AA AAAA AAFF
03B0   FFAA AAAA AAD6 FF2E  AE2E EEE2 FFFF 2CAB
03C0   BDBE 39FF 20EA 2AEA  2AFF FFFF FFFF FFFF
03D0   FF88 BA8A BA8A FFF8  F8F8 F8F8 F8F8 88D8
03E0   D8D8 D8F8 E000 0003
```

****

METEOR STORM

(00F0 - 0300)

T. GRAY, (11 yrs)
[redacted]

In this game you have to steer your space cruiser as far as you can through a meteor storm.

You get three space cruisers, which you steer around the meteors, which come flying at you randomly, using keys 1 (UP) and 9 (DOWN). The score is the number of meteors that you dodge.

For a harder game, change 0207 to 1A. For an easier game, change 022D to a higher value.

To change key functions, change 0219 for up and 021F for down.

```
00F0   EE7F EEAA 0055 00AA  2870 F870 2800 0506

0200   6300 6410 6503 6638  00E0 6810 6910 6A00
0210   A0F0 D343 D343 0232  6801 EBA1 74FF 6B09
0220   EBA1 7401 D343 7A01  4F01 1248 4A08 1278
0230   1214 CE01 FFC6 080C  A600 49A7 0009 5A26
0240   F78C 00FF 26EF 3900  A0F0 D343 A0F3 D345
0250   6C0F FC18 D345 75FF  4500 125E 1206 A0FD
0260   F033 F265 F029 00E0  D895 7804 F129 D895
0270   7804 F229 D895 1276  A0F8 D675 7001 6A00
0280   A0F0 C71B 6D02 FD15  FD07 3D00 1288 1214
```

****

WOODWORM

(0200 - 0300)

A. GROVES,
5 CAROLANNE CRT.,
MOOROOLBARK. 3138

A random forest of softwood and hardwood is displayed on the screen. A woodworm is shown as a dot on the LH edge of the forest. The object of the game is to eat your way through the forest to the RH side in the least number of moves. Use key 1 to move UP, 9 to move DOWN, 6 to move RIGHT.

You are penalised 1 point for each meal of softwood (the dots) and 6 points for each meal of hardwood (blank spaces). Your score is recorded in the TOP RH corner, and the best score in the current game series is shown in the BOTTOM RH corner. (Starts at 250 when program is loaded.) Press any key to start a new game.

Key functions can be found at the following locations. Move RIGHT, 0245. Move UP, 0249, Move DOWN, 024D.

```
0200   121C CB1F DAB5 3B01  1202 00EE 7A03 2202
0210   1222 6A02 CB1F A260  DAB1 1228 6A03 A261
0220   2202 3A2A 120C 1212  FA18 6500 2266 0000
0230   0000 22A0 0000 0000  4A2D 1292 6F00 FE0A
0240   A260 DAB1 4E06 7A01  4E01 7BFF 4E09 7B01
0250   DAB1 4A2D 1292 4F01  1282 690A F918 128A
0260   8080 E040 80E0 A22E  F533 6331 6402 F265
0270   F029 D345 7305 F129  D345 7305 F229 D345
0280   00EE 2266 7501 2266  1234 2266 7506 2266
0290   1234 6928 F918 22C6  1200 0000 0000 0000
02A0   A2E2 F755 A2E2 F865  6631 6717 A2EC F833
02B0   0000 F265 F029 D675  7605 F129 D675 7605
02C0   F229 D675 00EE 8D80  0000 6F00 8D55 0000
02D0   3F01 12DC 8850 0000  A2E2 F855 F00A 00E0
02E0   00EE 0000 003B 0200  3B17 7200 0101 0400
```

****

PRECISION DREAMS

L. DE VRIES,
VK3AXM,
[redacted]

Here is a group of three inter-related articles, two of which are programs to turn your DREAM 6800 into a precision measuring instrument, and the third an article on how to use them.

No.1 is a FREQUENCY COUNTER PROGRAM.
This turns PB1 of the DREAM into a frequency counter in the range 1 Hz to 10 KHz, with very good accuracy.
No.2 is a PERIOD TIMER PROGRAM.
This turns PB1 of the DREAM into a period timer for timing pulses (Negative or Positive going) in the range 50uS to 2550 uS with very good accuracy.
No.3 is an ARTICLE ON ADJUSTMENT OF TAPE INTERFACE.
This uses programs 1 and 2 above to adjust tones to 1200/2400 Hz, and Digital filter to 310 uS. This will allow all DREAM users to make all DREAMS compatible.

++++++++++++++

FREQUENCY COUNTER (0200 - 0300)

This program turns your DREAM into a frequency counter which will measure and display the frequency of a TTL level signal applied to PB1.
The range and acturacy is
20 Hz (± 0 Hz) - 10,000 Hz (± 10 Hz)

INSTRUCTIONS FOR USE:

1. Load program and start C000, FN, 3.
2. Apply signal to be measured to PB1 and press 0 or 1. Frequency will be displayed after 1 second.
3. Further measurements may be made by repeating step 2.

NOTE: a) By pressing 0 or 1, 1200 Hz or 2400 Hz respectively will be available at Pin 3 of IC24. (566)
b) If you do not want to hear the tone whilst the frequency is being displayed, change data at 0249 and 024F to 00 and 01 respectively.
c) There is a handy 16 bit Binary to BCD conversion routine at 028A - 02C6.

```
0200   F00A 6A00 6B00 00E0  4000 1212 4001 1216
0210   1200 0246 1218 024C  F029 2230 F129 2230
0220   F229 2230 F329 2230  F429 2230 1200 0000
0230   DAB5 7A05 00EE CE80  12C6 3BE7 01C6 7DE7
0240   00C6 37E7 0139 8DEE  C640 2004 8DE8 C641
0250   E700 0FCE 0283 DF80  8632 9720 CE00 00C6
0260   020E 9120 27FC F580  1226 FBF5 8012 27FB
0270   0896 202A F186 3FB7  8013 DF30 9630 D631
0280   8D08 397A 0020 7D80  123B CE00 30DF 3CCE
0290   02BD 7F00 3BE0 01A2  0025 057C 003B 20F5
02A0   EB01 A900 36DF 3EDE  3C96 3B8B 30A7 0032
02B0   08DF 3CDE 3E08 088C  02C7 26D6 3927 1003
02C0   E800 6400 0A00 0100
```

++++++++++++++

PRECISION DREAMS (Cont)

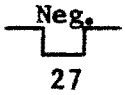
PULSE PERIOD TIMER (0200 - 0250)

This program turns your DREAM into a pulse timer which will time a positive or negative going pulse in the range 50 - 2550 uS (± 10 uS), applied to PB1.

INSTRUCTIONS FOR USE:

1. Load program and start C000, FN, 3.
2. Apply signal to PB1 and press any key.
3. For further measurements press any key.

NOTE: Change the following data for Negative or Positive going pulses.



	Neg.	Pos.
023B	27	26
0241	26	27
0247	27	26

```
0200   F00A 6A00 6B00 00E0   0228 A030 F333 6300
0210   F029 2222 F129 2222   F229 2222 F329 2222
0220   1200 DAB5 7A05 00EE   CE80 12C6 38E7 01C6
0230   7D86 34BD C307 C602   F580 1227 FB4F F580
0240   1226 FB4C F580 1227   FA97 33C6 3FE7 0139
```

++++++++ ++++++++

STANDARDISED ALIGNMENT OF TAPE INTERFACES

Wouldn't it be nice if all our DREAMs were aligned so that we could swap programs and not have to worry about twiddling to get a successful load - interested? Read on.

To make this even more attractive I will tell you that you do not need any test equipment of any kind except for the DREAM itself with some suitable programs. These have just been described to you.

1) ADJUSTMENT OF 1200/2400 Hz TONES.

1. Load and run Frequency Counter program.
2. Using a short jumper lead, connect Pin 3 of IC24 (566) to PB1.
3. Press 0 and note frequency.
4. Press 1 and note frequency.
5. Change 5.6 K resistor until tones are 1200/2400 ± 10 Hz. Some trimming may be required by series and parrallel combination of resistors.
6. Repeat steps 3 to 5 until satisfactory results have been obtained.

2) ADJUSTMENT OF DIGITAL FILTER.

1. Load and run Pulse Period Timer. (-VE going.)
2. Connect Pin 1 of IC22 to PB1.
3. Feed some data from tape into DREAM and press any key.
4. Note period in uSecs.
5. Adjust trimpot (5K) and repeat steps 3 to 5 until 310 uS has been achieved.

HOW ABOUT THAT!

Because all our DREAMs run at 1 MHz, our interfaces have been adjusted using the same standard, which is not something we could claim if we had used Frequency Counters and C.R.O.'s to do the job.

****

CROSS (0080 - 0400)

GRAEME V. SAMWAYS,
[redacted]

Remove as many of the checkers as possible by jumping over them either horizontally or verticaly.

Enter the Horizontal location 1 - 7, followed by the Vertical location, 1 - 7, then the direction of Travel, 5 Up, 8 Left, A Right, D Down.

The square you jump to must be empty, and there must be one checker between it and the one you jumped from.

The bottom score is the number of checkers left, the top one is the lowest score achieved in the current game series.

```
0080    7505 23C8 4E01 1264   7505 23C8 4E00 1264
0090    A3DA DAB3 7A05 DAB3   7A05 DAB3 A0F4 F265
00A0    6A30 6B14 23F0 A0F4   74FF 4401 10F8 F433
00B0    F265 6A30 6B14 23F0   125C

0200    6320 6420 8030 8045   3F00 8340 6A30 6B04
0210    A0F0 F333 F265 23F0   6420 6A30 6B14 A0F4
0220    F433 F265 23F0 6A0A   6B02 23E0 6A0A 6B06
0230    23E0 6A00 6B0A 60FC   23E2 6A00 6B0E 23E0
0240    A3D8 DAB2 7A05 23E0   6A00 6B12 60FC 23E2
0250    6A0A 6B16 23E0 6A0A   6B1A 23E0 FA0A 6008
0260    3A00 1268 F018 125C   80A2 3000 1264 80A0
0270    8004 8004 8A04 7AFB   FB0A 6008 4B00 1264
0280    80B2 3000 1264 8BB4   8BB4 7BFE 4A0A 12A8
0290    4A0F 12A8 4A14 12A8   4B0A 12A8 4B0E 12A8
02A0    4B12 12A8 6020 1264   A3D8 DAB2 6010 13D0
02B0    F018 FC0A 3C05 12D6   4B02 12B0 4B06 12B0
02C0    3B0A 12D2 4A0A 1340   4A0F 1340 4A14 1340
02D0    12B0 4B0E 12C4 3C0D   12F8 4B16 12B0 4B1A
02E0    12B0 3B0E 12F4 4A0A   1340 4A0F 1340 4A14
02F0    1340 12B0 4B12 12E6   3C08 131A 4A00 12B0

0300    4A05 12B0 3A0A 1316   4B0A 1340 4B0E 1340
0310    4B12 1340 12B0 4A0F   1308 3C0A 1340 4A19
0320    12B0 4A1E 12B0 3A14   133C 4B02 12B0 4B06
0330    12B0 4B16 12B0 4B1A   12B0 1340 4A0F 132A
0340    85A0 86B0 A3D8 DAB2   4C05 1378 4C08 1398
0350    4C0A 1080 13B8 0000   7604 23C8 4E01 1264
0360    7604 23C8 4E00 1264   A3DA DAB3 7B04 DAB3
0370    7B04 DAB3 109C 0000   76FC 23C8 4E01 1264
0380    76FC 23C8 4E00 1264   A3DA DAB3 7BFC DAB3
0390    7BFC DAB3 109C 0000   75FB 23C8 4E01 1264
03A0    75FB 23C8 4E00 1264   A3DA DAB3 7AFB DAB3
03B0    7AFB DAB3 109C 0000   4C0D 1358 3C0F 1264
03C0    6730 F718 00E0 1204   D562 8EF0 D562 00EE
03D0    3F01 12B2 DAB2 1264   0060 F0F0 F090 F000
03E0    6000 A3DC DAB3 7A05   7001 3003 13E4 00EE
03F0    F129 DAB5 7A04 F229   DAB5 7AFC 00EE
```

****

TOWERS OF HANOI

(0080 - 0400)

RUSSELL YANKUS,
[redacted]

When the game starts, you will be prompted to enter a number between 2 and 7. This number represents the number of discs that will be placed onto stack #1, shown on the LHS of the display. Each disc is of a different diameter, with the smallest at the top.

The object of the exercise is to move all of the disks to stack # 3 shown on the RHS of the display, in such a manner that no disk ever lies on top of another disc that is smaller. The middle stack (# 2) is used to temporarily store discs during moving.

Keys 1, 2 & 3 are used to represent the stack numbers. You may move the discs in any order. Illegal moves are detected and your score is advanced by 1. The minimum number of moves is calculated by 2 to nth power -1, where n is the number of discs selected.

Enter the program, save it, then run from 0080. I.E.0080, FN, 3, NOT C000, FN, 3. Select a key between 2 & 7 and hold it down until the discs are displayed. When you have successfully moved all of the discs to the RHS stack, the number of moves you took will be displayed. For another game, hit RST, 0080, FN, 3.

```
0080   4FCE 0140 A700 088C  0200 26F8 CE02 7E7E
0090   C005 A158 6201 6380  2248 A168 6203 63C0
00A0   2248 6207 63E0 A178  2248 620F 63F0 A188
00B0   2248 621F 63F8 A198  2248 623F 63FC A1A8
00C0   2248 627F 63FE A1B8  2248 6001 6A06 6B1A
00D0   F029 DAB5 7001 2238  4004 10E0 7A18 10D0
00E0   A0E8 6000 F055 12DA  0F00 0105 F265 F029
00F0   DAB5 F129 7A04 DAB5  F229 7A04 DAB5 1238

0200   6D0E 6C01 F165 3000  00EE 7C01 4C08 00EE
0210   FD1E 1204 8200 8310  6000 6100 6401 6D06
0220   F155 FD1E F155 8C45  FD1E 4C00 1230 1220
0230   8020 8130 6C01 00EE  6602 F618 6610 F615
0240   F607 3600 1240 00EE  6D10 8C45 4C00 1254
0250   FD1E 124A 2230 121E  7700 7706 7353 0000
0260   5500 5405 2454 0700  5500 5605 2262 0000
0270   5500 5405 2151 0700  5720 7406 7656 A258
0280   FD65 A180 FD55 A266  FD65 A18E FD55 A274
0290   F965 A19C F955 6A3C  6B10 6002 F029 DAB5
02A0   2238 DAB5 E0A1 12B0  7001 3008 129C 129A
02B0   8E00 A140 6101 F065  F055 7101 31C8 12B8
02C0   6C01 6401 4E07 1092  4E06 109A 4E05 10A2
02D0   4E04 10AA 4E03 10B2  10BA F80A 4801 12EE
02E0   4802 1300 4803 1312  6610 223A 12DA A158
02F0   2200 4C08 12E8 A158  2214 A140 2220 1348

0300   A15B 2200 4C08 12E8  A15B 2214 A143 2220
0310   1354 A15E 2200 4C08  12E8 A15E 2214 A146
0320   2220 1360 F90A 4901  1338 4902 1338 4903
0330   1338 6610 223A 1324  A0E8 F065 7001 A0E8
0340   F055 F033 00EE 1338  2324 8020 8130 A140
0350   2214 136C 2324 8020  8130 A143 2214 136C
0360   2324 8020 8130 A146  2214 136C 4901 1378
0370   4902 138A 4903 1396  A158 2200 0000 8C45
0380   A158 2248 13A2 13EC  12DA A15B 2200 0000
0390   8C45 A15B 1382 A15E  2200 0000 8C45 A15E
03A0   1382 A158 23BC 3F01  13D4 A15B 23BC 3F01
```

(See Page 17 for 03B0 - 0400 and 0100 - 0200)

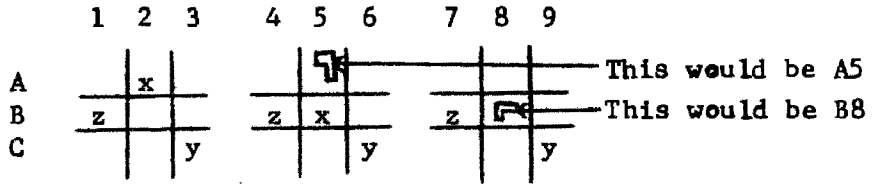
THREE DIMENSIONAL NOUGHTS AND CROSSES

(0200 - 0300)

D. WOOLNOUGH,
[redacted]

This is simply three games of noughts and crosses side by side, requiring one symbol in each of the formations in symmetrical order for a win. Because three games have to fit side by side in the screen space available, I have used differing but similar symbols to indicate which player is which, but not full noughts and crosses.

The means of entering a symbol in a particular square is shown below. If desired, the three games can be played individually.



The same symbols at the 'x' position for example would constitute a win.The same for 'y's and 'z's.

```
0200   6C02 A29A 6A06 6B0D  228E A2A2 6A0D 228E
0210   A2AA 6A06 7B07 228E  A2B2 6A0D 228E 6A1D
0220   6B03 3C01 12BA 12C0  FD0A 3D0A 1232 6D0E
0230   1240 3D0B 123A 6D13  1240 3D0C 1228 6D18
0240   FE0A 3E01 124A 6E07  1288 3E02 1252 6E0C
0250   1288 3E03 125A 6E11  1288 3E04 1262 6E1A
0260   1288 3E05 126A 6E1F  1288 3E06 1272 6E24
0270   1288 3E07 127A 6E2D  1288 3E08 1282 6E32
0280   1288 3E09 1228 6E37  3C01 12D2 12E4 DAB7
0290   7A13 DAB7 7A13 DAB7  00EE 0808 0808 FE08
02A0   0800 2020 2020 FE20  2000 0808 FE08 0808
02B0   0800 2020 FE20 2020  2000 A2C6 DAB5 1228
02C0   A2CC DAB5 1228 2050  88F8 8800 F088 F088
02D0   F000 A2F0 DED2 3F01  6C01 A2C6 DAB5 6110
02E0   F118 1222 A2F2 DED2  3F01 7C01 A2CC 12DC
02F0   C080 40C0
```

****

TOWERS OF HANOI (Cont)

```
03B0   13DA A15E 23BC 3F01  13E0 1386 2200 4C08
03C0   13D0 4C07 13D0 8A00  2210 6F00 80A5 00EE
03D0   6F01 00EE 6801 23E6  12EE 6802 23E6 1300
03E0   6803 23E6 1312 6630  223A 00EE A15E 2200
03F0   8CE4 3C08 12DA 6A20  6B10 A0E9 20EC 13EC

0100   0000 0000 0000 0000  7755 DCC1 DC15 DDDC
0110   2555 1501 5015 5548  2555 DC81 581D D548
0120   2555 1841 5015 5548  2729 D581 D015 55DC
0130   0000 0000 0000 0000  0000 0000 0000 0000
0140   0000 0000 0000 0000  6501 D4CC 15DD 54C0
0150   5521 5510 1555 5500  6701 D488 1DD5 9480
0160   5221 9444 0955 5440  6201 5D98 0955 5D80
0170   0000 0000 0000 0000  0000 0000 0000 0000
0180   FFFF FFFF FFFF FFFF  8888 F88F 8F88 F88F
0190   AAAA FBAF EFDA FBAF  AA8A F9AF 8FDA FAAF
01A0   AAAA FBAF EFDA FAAF  8888 FBAF 8FD8 F88F
01B0   FFFF FFFF FFFF FFFF  0000 0000 0000 0000
01C0   0000 0000 0000 0000
```

****

LOTTO

(0200 - 0400)

TONY HORNCASTLE,
[redacted]

Yet another LOTTO program. This time the coupon is displayed and the chosen numbers high-lighted.

Key A to select numbers in each game.

```
0200   231A 6E0A CDFF EE9E  1204 22B4 6A18 6B08
0210   8080 3800 22DE 7801  6A18 8080 22DE 6700
0220   6628 86C5 228E 86D4  224C A090 F71E F71E
0230   F155 22F4 80D0 8CD0  A080 F71E F055 22DE
0240   7701 6904 F918 3706  1220 1202 A08A F165
0250   8A00 8B10 7A04 3A3F  1260 6A27 7B04 126C
0260   3B1A 126C 3A37 126C  6A27 6B02 A3A4 DAB3
0270   76FF 3600 1280 A08A  80A0 81B0 F155 00EE
0280   6903 F915 F907 3900  1284 DAB3 1254 CD3F
0290   6928 8D95 3F00 128E  7D29 A080 F465 9D00
02A0   128E 9D10 128E 9D20  128E 9D30 128E 9D40
02B0   128E 00EE 6700 22F4  A080 F71E F065 3800
02C0   22DE 6000 F055 4800  12D6 A090 F71E F71E
02D0   F165 A3A4 D013 7701  3706 12B6 00EE A086
02E0   F033 A087 F165 F029  3000 DAB5 7A04 F129
02F0   DAB5 00EE 8070 8074  8074 8074 6B13 8300

0300   6A00 00EE 6A0C 00EE  6A18 00EE 6A00 1316
0310   6A0C 1316 6A18 7B07  00EE A39C 6A26 6B01
0320   DAB1 7B01 3B1E 1320  7A04 3A42 131E A386
0330   6B01 6A27 DAB1 7A08  3A3F 1334 7B04 3B21
0340   1332 A3A0 6A37 6B1A  DAB4 6A08 6B01 A382
0350   DAB5 A387 7A08 DAB5  A38C 7A08 DAB5 6A00
0360   6B08 A391 DAB5 A396  7A08 DAB5 A39B 7A08
0370   DAB5 6800 A08A 6033  611A F155 6C00 00EE
0380   0000 8E8A 8A8A EEEE  4444 4444 E0A0 A0A0
0390   E0F7 84B7 94F4 BEAA  AAAA A2F0 80E4 80F4
03A0   1111 11FF A040 A000
```

****

DRAW

(0200 - 0300)

B. KIDDIER,
[redacted]

This program will hold two drawings which can be exchanged and modified any number of times.

Press 8 for LEFT, A for RIGHT, 5 for UP, and D for DOWN. If you want to restart your drawing, press 0. If you want to save an 'on-screen' drawing on cassette, press E to store it, and then dump 0300-0400 onto tape.

```
0200   6D00 6E00 A292 DDE1  6D3F DDE1 6E1F DDE1
0210   6D00 DDE1 6D20 6E10  DDE1 6C05 ECA1 1242
0220   6C08 ECA1 1246 6C0D  ECA1 124A 6C0A ECA1
0230   124E 6C0E ECA1 125C  6C00 EC9E 121A 00E0
0240   1200 7EFF 1250 7DFF  1250 7E01 1250 7D01
0250   6C0A FC15 FC07 3C00  1254 1218 6A00 A100
0260   FA1E F765 A080 F755  A300 FA1E F765 A090
0270   F755 A080 F765 A300  FA1E F755 A090 F765
0280   A100 FA1E F755 7A08  3A00 125E A292 FC18
0290   121A 8080
```

****

CONTENTS

ED. FARRELL,
[redacted]

NEWSLETTER NUMBER ELEVEN, JULY, 1981

NEWSLETTER NUMBER TWELVE, AUGUST, 1981

NEWSLETTER NUMBER TWELVE, AUGUST, 1981. (Cont)

NEWSLETTER NUMBER THIRTEEN, SEPTEMBER, 1981

****