4th article gives the lowdown on

Chip-8 programming for the DREAM 6800 computer

In this fourth article on the DREAM 6800 the author gives hints on CHIP-8 programming. Also featured is a substitute circuit for the 6875 clock chip using low cost TTL devices and the full size artwork for the PCB. A future article will give details of RAM expansion.

by MICHAEL J. BAUER

After a while, when the provided video games become a bit of a yawn, you will want to write your own programs. There is no language as powerful as CHIP-8 which can be learned with such ease. The function of most of the instructions can be understood from the table, but some need further explanation. First, it might be an idea to re-read the CHIP-8 summary given in the May article.

The display instruction (DXYN) is the most important. It treats the screen as a coordinate grid of dots, numbered from 0 to 63 (00-3F hex) from left to right across the screen, and from 0 to 31 (00-1F hex) from top to bottom. Two variables of your choice are used to specify the coordinates of a symbol to be displayed. The symbol may be any size up to 8 dots across by 16 dots down. Larger symbols may be shown by using more than one DXYN instruction, possibly in a loop. Various symbols are defined by making up a pattern of bytes and storing this data along with the program. As an example, let us say we want to show an "X", 7 x 7 dots in size. Thus, N is 7. The screen coordinates we will choose to be variables VA and VB, i.e. X=A and Y=B. Thus the instruction will be DAB7. But how does the interpreter know where to find our symbol pattern? A special index variable, called "I", can be set to point to anywhere in the bottom 4k of memory, using an AMMM instruction. Let us put our pattern at location 0210 onwards, thus:-

Address	Binary Data	Hex Data
0210	1000 0010	82
0211	0100 0100	44
0212	0010 1000	28
0213	0001 0000	10
0214	0010 1000	28
0215	0100 0100	44
0216	1000 0010	82

To display this pattern in the upper left hand corner of the screen, we would initialize variables VA and VB to zero, and set I=210. Note, if N=0, a 16 byte pattern will result. The program, with comments, is shown below:-

```
0200   6A00   VA=00             Set coordinates
0202   6B00   VB=00
0204   A210   I=210             Set pointer
0206   DAB7   SHOW 7@VA,VB      Show 7-byte pattern
0208   F000   STOP              Jump to monitor
020A
020C
020E
0210   8244   DATA              Pattern for "X"
0212   2810
0214   2844
0216   8200
```

Note that the first CHIP-8 instruction must be at 0200. The program is executed by a GO from C000, which is the interpreter's starting address. Try setting VA and VB to different starting values, then re-run the program. Note that these values specify the position of the upper LH corner of the symbol.

An important feature of the SHOW instruction is that if a symbol is displayed and it overlaps another symbol already there, then the overlapping spots are erased and variable VF (the "flag" variable) is set to 01. If no overlap, VF=00. This feature can be used to erase a symbol, by showing it again at the same coordinates, without erasing the whole screen (which can be done with a 00E0). Of course, you have to keep track of the positions of each different symbol used in this way. Variable VF can be used to see if two objects collided, in an animated game. An object can be made to move about on the screen by erasing it and re-showing it in a new position each time. This program illustrates:-

```
0200  6A00  VA=00
0202  6B00  VB=00
0204  A210  I=210
0206  DAB7  SHOW 7@VA,VB
0208  DAB7  SHOW 7@VA,VB
020A  7A01  VA=VA+01
020C  7B02  VB=VB+02
020E  1206  GOTO 206
0210  8244  DATA
0212  2810
0214  2844
0216  8200
```

The speed and direction of motion can be manipulated with the instructions at 20A and 20C by changing the values which are added. Note that adding FF is the same as subtracting 01; (refer to a text on "two's complement" arithmetic). The motion can be slowed down by putting a time delay inside the loop.

The random byte generator in CHIPOS is unique, and achieves longer sequences and higher randomisation than conventional software pseudo-random sequencers by utilising the fact the program bytes are "kind-of" random. In a VX=RND.KK instruction (CXKK), a variable is set to a random value which has been masked by (i.e. ANDed with) a constant (KK). Thus, random numbers covering a specified range, and falling into precise intervals, can be selected. For example, a C01E instruction will give only even numbers in the range 0 to 30 (00-1E hex).

CHIPOS has built-in patterns for the symbols 0 to 9 and A to F, and CHIP-8 provides an instruction to allow you to display the contents of any variable as a hex digit. Only the least significant 4 bits are relevant. For example, to show the hex value of V6, we would use an F629 followed by a DXY5, where X and Y are again arbitrary. The F629 sets up I to point to the symbol corresponding to the value of V6 (LSD).

Another useful instruction, FX33, lets you find the 3-digit decimal equivalent of any variable; e.g: F433 would store 3 bytes in memory, at the location specified in I. To display this 3-digit number, you will need to be familiar with another pair of instructions: FX55 and FX65.

FX55 takes the values of variables V0 up to VX (incl.) and stores them in successive memory locations, indexed by I. FX65 does the reverse, i.e. re-loads the variables from memory. These powerful instructions not only extend the number of available variables, but also let you perform array processing. Note that the pointer (I) auto-increments with these two instructions, i.e. I is advanced by the number of variables stored or loaded (X+1), provided that a page boundary is not crossed. (A "page" is 256 bytes.) Also note that if X=0, only one variable (VO) is affected.

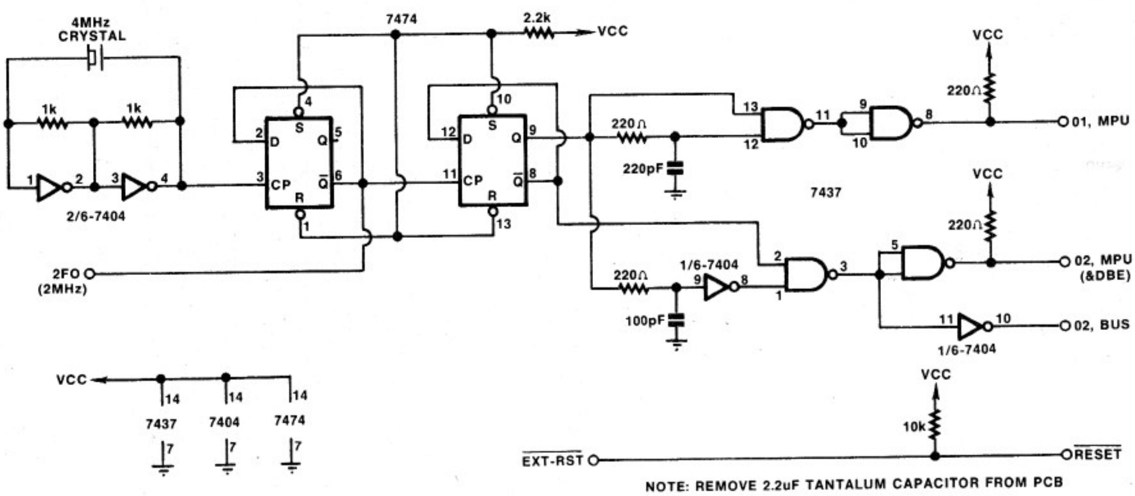
We can therefore use F256 to load variables V0, V1 and V2 from memory at I. If we had previously used an F433 to store the 3-digit decimal equivalent of V4 in memory at I, then V0, V1 and V2 would now contain the "hundreds", "tens", and "units" (resp.) of the value of V4. These can be displayed with the FX29 and DXYN instructions, as explained. While this may be confusing at first, it makes for a very versatile language, as you will come to appreciate.

A small part of a program, called a subroutine, can be accessed several times from different parts of a larger program. Further, a subroutine can "call" other subroutines (known as subroutine nesting). Each subroutine must end with a RETURN statement (00EE) so that, upon completion, control will return to the instruction following the particular "DO" statement (2MMM) that called it.

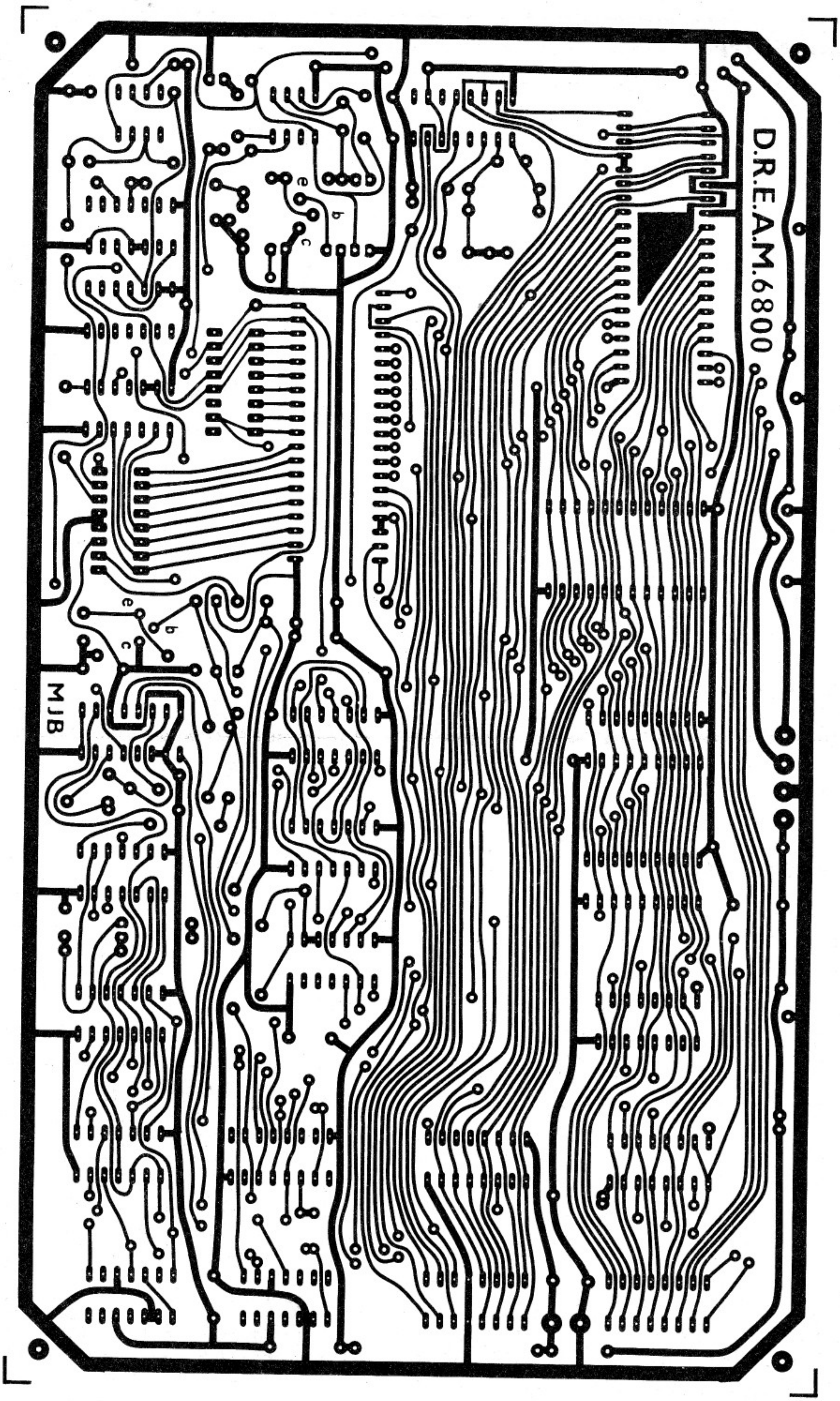
Everything discussed in the last few paragraphs is illustrated in the following program, plus the use of the timer and tone instructions, so you can see how easy it is. Have a close look, and try to understand its workings. The program counts down V3, converting it to decimal, showing it, and bleeping, every second.

0200	63FA	V3=FA	start counter at 250
0202	A240	I=240	point to workspace
0204	F333	MI=DEQ,V3	store dec. eq. of V3
0206	F265	VO:V2=MI	Load same into V0:V2
0208	6418	V4=18	set display position
021A	6510	V5=10	
021C	F029	I=DSP,V0	display V2
021E	222C	DO 22C	
0210	F129	I=DSP,V1	display V0
0212	222C	DO 22C	
0214	F229	I=DSP,V2	display V1
0216	222C	DO 22C	
0218	6602	V6=02	bleep for 2x20 msec
021A	F618	TONE=V6	
021C	6630	V6=30	wait for 48x20 msec
021E	F615	TIME=V6	(total 1 sec)
0220	F607	V6=TIME	check timer
0222	3600	SKF V6=00	
0224	1220	GO TO 220	
0226	73FF	V3=V3+FF	decrement counter
0228	00E0	ERASE	clear screen
022A	1202	GO TO 202	repeat . . .
022C	D455	SHOW 5@V4,V5	subr. to show digit
022E	7404	V4=V4+04	move "cursor" right
0230	00EE	RETURN	

Here is a substitute circuit for the 6875 clock chip:



Just before the July issue was due to be run on the presses, the shortage of 6875 clock chips became apparent. It seems that it could be several months before Motorola, Inc, USA is able to restore supplies. In the meantime, designer M.J. Bauer has produced a substitute circuit for the 6875 using cheap and readily available TTL ICs. This circuit may be built up on a small section of Veroboard and linked to the DREAM PCB via a ribbon cable fitted with DIL plug (16-pin) and IC socket, in the 6875 position. When the time comes, the TTL circuit can be discarded and the 6875 plugged in, instead. Note that the reset circuitry must also be changed slightly, as noted on the circuit above.



Here is the full size artwork for the printed circuit board.

While the above program serves to illustrate some of the trickier CHIP-8 statements, it is not a good example of the power and efficiency of the language. To see that, one has to analyse a more complex, graphics oriented program, such as an animated video game. It is good experience to "dis-assemble" one or more of the games provided, to see how the programmer tackled the problem. You should therefore deduce: which numbers are instructions and which are data; what each variable is used for; and what is stored in various memory workspaces; etc. (Kaleidoscope and TV-Typewriter not recommended for starters.) Flowcharting is also a handy programming tool that will increase your expertise.

I have presented only a very sketchy description of how to write programs. A lot of practical experience is the only way to learn and become proficient. Test the operation of each of the instructions in a short routine, so that its operation becomes clear. Before attempting any complex video games, try some of these simpler exercises:-

1. A program that waits for a key depression, then displays the corresponding hex digit on the screen. (Looping indefinitely.)
2. Same as (1), but rejects keys A to F by returning to monitor.
3. Show an 8 x 8 symbol of your choice on the screen and make it move left when key 4 is held down and right when key 6 is held (using EX9E or EXA1).
4. Make the above 8 x 8 symbol move randomly about the screen.
5. Program the game of NIM. Show 21 objects on the screen. Two players take turns to remove 1, 2 or 3 objects. Player to take last object(s) wins.
6. Imagine a 4 x 4 square game board. The keypad is also a 4 x 4 matrix. Program accepts a hex key, then places a symbol in corresponding position on screen.
7. As above, but alternating between two different symbols.
8. Invent a two-player game based on the above principle, and program your computer to win against a human opponent.

Once you can do the above, you're ready for Lunar Lander,LIFE, Blackjack, and other favourites. Add a 2k RAM board and you can try for CHESS or STAR-TREK.

APPENDIX: HEXADECIMAL

There's nothing complicated about it, but it might help if you had 8 fingers on each hand. Then you could count from 0 to 15 (instead of 0 to 9) before having to use carry. HEX is convenient because each digit can be represented by exactly 4 binary digits (bits), without having any missing codes or extraneous codes:-

Decimal	Binary	HEX
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

The symbols A to F are used to denote the numerals 10 to 15. Furthermore, 4 divides into 8 exactly; so you can represent an 8-bit binary number with 2 hex digits, without having any bits left over; unlike the OCTAL (base 8) system, which has had many programmers pulling out their hair! Thus we can easily convert between binary and hex, simply by grouping bits into fours: e.g.:-

What is 26F0 in binary?

2 6 F 0

Answer = 0010 0110 1111 0000 (from above table)

What is 01111100 in hex?

0111 1100

Answer = 7 E

As well, 16=4x4, and 4+4=8, and PIAs have 8-bit ports, which makes 16-key keypads ideally suited. So HEX is very convenient all round, and easy to master once you memorize the above table!

EDITOR'S NOTE: Reaction to the DREAM 6800 articles has been unprecedented and it seems that a very large number of readers intend building this circuit. Unfortunately, there have been component shortages, including the 6875 and the 2708 EPROM. But it now seems (at time of writing, June 26) that most of these problems are close to solution.

We are informed that programmed 2708 EPROMs containing CHIPOS are now available from Silicon Valley stores and from All Electronic Components (formerly E.D. & E. Sales Pty Ltd), 118 Lonsdale Street, Melbourne, Victoria. As well, complete kits for the DREAM 6800 are available from J.R. Components, PO Box 128, Eastwood, NSW 2122.