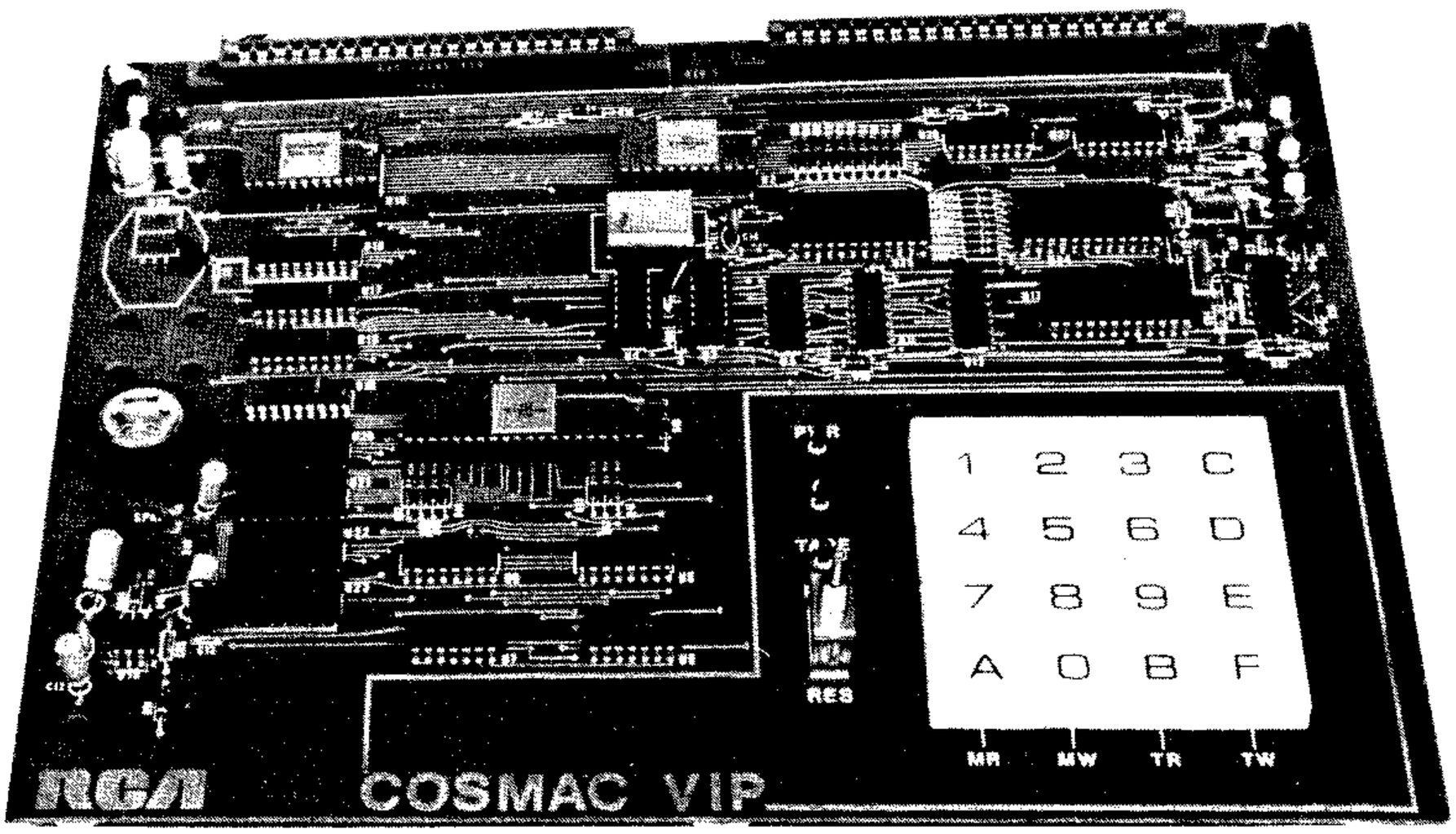
VIPER

VOLUME 1 JUNE 1978 ISSUE 1



$2.00

AN ARESCO PUBLICATION

EDITORIAL

We are pleased and proud to present yet another innovative product: the VIPER.

The market is flooded with computer-oriented magazines and newsletters. There are publications which cater to the hardware buff and there are others which handle only the software end of things. Then there are user notes and newsletters and bulletins. The VIPER is different. It will be all of these things, and will be dedicated to the support of the COSMAC VIP, the low-cost RCA computer designed with the home hobbyist in mind.

We hope to provide VIP owners with a 'balanced diet' of information about hardware and software, and to provide a forum for information exchange among VIP users.

You'll get the latest and most up-to-date info about VIP products, peripherals, and accessories we (and anyone else) are planning. You'll read here, before anywhere else, all the latest news about software; new programs, new concepts, new applications.

Many of the articles you find in these pages will have been submitted by veteran VIP users. Some of the articles will be dedicated to newcomers to the world of computers. We'll list and describe all the VIP clubs we know about as they are formed.

We hope **you** will drop us a line or two, sharing **your** VIP experiences. You don't have to be an expert to exchange ideas and information; you simply have to be interested.

The design of the RCA COSMAC VIP was based on the concept of maximum return for minimum investment. As you browse through these pages, you'll find that the VIPER carries this theme one step further. We'll tell you where to find low-cost parts to build your own peripherals, We'll let you know when and where you can buy new RCA products (and how much they cost and how well they work). We'll share techniques and tricks of programming and applications that will save you time and money without sacrificing so much as a second of the fun or the RCA quality you're accustomed to.

The VIP and the VIPER are designed for **you**. If you particularly like or dislike something about the system, the software, or the newsletter, let us know. If you have questions, comments, praise or peeves, let us know that, too. We hope to establish a base of communications between RCA and the VIP user; between the VIP user and the other VIP users; and between the new VIP owner and the rest of the VIP world. To do that, we need **you** to communicate with us. What do **you** want to know? What are **you** puzzled about or interested in? What have **you** done with **your** VIP? If you'll share **your** experiences and observations with us, the VIPER will be the most widely read newsletter in existence!

SUBSCRIPTION RATES, ADVERTISING RATES AND OTHER ESSENTIAL INFORMATION

The VIPER is published ten times per year and mailed to subscribers on the 15th day of each month except June and December. Single copy price is $2.00 per issue, subscription price is $15.00 per year (all ten issues of one volume.) Dealer prices upon request. Outside of Continental U.S. and Canada, add $10.00 per subscription for postage ($1.00 for single copy).

Readers are encouraged to submit articles of general interest to VIP owners. Material submitted will be considered free of copyright restrictions and should be submitted by the 1st day of the month in which publication is desired. Non-profit organizations (i.e., computer clubs) may reprint any part of the VIPER without express permission, provided appropriate credit is given with the reprint. Any other persons or organization should contact the editor for permission to reprint VIPER material.

Advertising rates are as follows:

1/4 page — $50
1/2 page — $90
3/4 page — $130
full page — $160

Less than 30% of the VIPER will be available for advertising. Please send camera ready copy in the exact page size of your ad on 8-1/2 x 11 white stock by the 1st day of the month in which you'd like the ad to appear. Photos should be glossy black & white in the exact size to be printed. Payment required with copy.

The VIPER is an Aresco Publication, edited by Terry L. Laudereau. For information contact Editor, VIPER, P.O. Box 43, Audubon, PA 19407.

This issue of the VIPER has been sponsored by RCA, without whose help and encouragement publication would have been long delayed. However, the VIPER is not associated with RCA in any way, and RCA is not responsible for its content.

VIPWORLD

A Brief History of the COSMAC VIP

About five years ago, Joe Weisbecker, the father of the VIP, developed a system he called FRED (Flexible Recreational and Educational Device). FRED didn't do data processing (any other computer could do that) or number crunching. FRED played games and experimented around with Computer Aided Instruction.

As FRED grew and matured, it became clear that everyone could have a FRED at some ridiculously low cost—and that the opportunity should be offered to the world at large. The VIP is a direct descendent of FRED, with all of FRED's capability and then some. The VIP was designed with the home hobbyist and the scholastic budget in mind. Video games and CAI are important to VIP; file manipulation and data processing and number crunching and compiler problems were not. Computers can be fun, RCA reasoned, if you don't have to wade through an operating system and a complicated 'high-level' language like COBOL. People don't need to be hard-core computer freaks to like computers, if computers aren't hard-core data freaks.

And so the VIP was created. The only computer in the world (that you can program) that is designed primarily to entertain you. As new applications are discovered for the VIP, we'll print reports for you—and new uses surface every day. It is obvious that graphic games only scratch the surface of the VIP's capabilities.

VIP OWNERS UNITE!

Ever wondered whether anyone else in your area owns a VIP? If there is another VIP owner near you, what is he/she doing with his/her VIP? How can you find out about other VIP owners?

You can start by joining or forming a VIP User Group. Let us know who you are and where you are, and we'll let you know who owns a VIP in your area. When you get your group going, we'll publish all the user notes we can, to let other groups know what you're discovering, planning, and putting together. We'll begin a Software Exchange library so you can have access to programs written by other VIPers in other parts of the country, and we'll let you know when and where other groups are meeting so you can visit if you happen to be in their area.

If you aren't sure you want to join or form a VIP user group, don't let that keep you from sending us your new ideas and designs! Don't hesitate to take advantage of the VIPER, the software library, or the information exchange. You're entitled to all of it, since it's all designed with **you** in mind!

To our knowledge, there are only two COSMAC VIP User Groups, and both of them are subgroups of larger Computer Clubs. ACGNJ (The Amateur Computer Group of New Jersey) and ACE (Association of Computer Experimenters) in Ontario, Canada. We're hoping to reprint material from their respective newsletters in forthcoming issues, so don't go 'way. . . .

The VIPER is not associated with RCA in any way. We just share an appreciation for the VIP.

When you toggle the 'reset' switch while pressing the C key, a bunch of garbage appears on your screen. Not to worry. . . that's only the stack running around in circles, and is perfectly normal. . . .

FOCUS

Video Display Tips

RF video modulators pose a special problem known as spurious radiation. It is caused by harmonics of the primary frequency 'leaking' into the air waves, causing interference on televisions and radios in your home and in other, adjacent homes. You can check for such interference by turning on another TV while your VIP and modulator are running. Tune the TV through all 12 VHF channels (2 – 13) and listen for static noise and watch for interference with the quality of the picture.

If you are causing interference by spurious radiation leaks, the FCC (Federal Communications Commission) will no doubt receive a complaint from your neighbors, and you will be told to either correct the problem immediately (if not sooner) or stop using the modulator.

The best way to correct the problem (since you don't want to stop using the modulator) is to completely enclose the device in an all-metal box. Use only coaxial cable for connecting the VIP to the modulator and the modulator to the TV. Where possible, use BNC or SO-239 coaxial connectors, and only operate the unit at or below the specified supply voltages.

The "Waterloo Modulator" described in the January, 1978 issue of **Byte** magazine is ideal for hot chassis, transformerless televisions. Most computer stores will be able to get a back issue for you if you don't have a copy.

PULSE

Memory for Your VIP

As of March 1st, 1978, RCA Solid State Division began selling the first VIP Kit Option Package. It contained four 4K bit RAMs, I/O devices, and an expansion socket.

A number of users had contacted RCA trying to find a better source of RAM for their VIPs, because they weren't willing to wait for the quoted delivery schedule of 6 months to a year. And, because RCA is determined to properly support its products, the Solid State division announced two VIP Option Packages.

The option best suited to your VIP will depend on the type of RAM included in your specific kit. If you have the AMD9131 (22-pin, 4K bit) static ram, VIP Option package CDP18S731 is the one you want. If you have Intel's 2114 or the TI 4045 (they are equivalent 18-pin, 4K bit static RAMs), the Kit Option number is CDP18S745.

These kits will fill out the onboard RAM space and complete the I/O port on the VIP. You might want to buy two kits, putting one set of RAMs on the VIP main board, and the other on the expansion interface for I/O, saving the extra board for projects on the expansion port.

To order either Kit Option Package, contact either the store or distributor from whom you bought your VIP, or send a check for $69.00 (plus $1.50 for postage & handling) and appropriate state sales tax to VIP Kit Option Package, Box 3200, RCA Solid State Division, Somerville, NJ 08876.

SOFTWARE CORNER

Tape Read/Write Routines

Many VIP users have requested code in the operating system that reads and writes to tape. The following two routines are initialized in the same manner. In the operation of the routines, R(E.Ø) will contain the pages to be transferred (up to FF pages). R(6.1) will contain the page of memory to start the transfer (1-FF), and R(3.1) is set to the page on which the subroutine resides. Do not change any of the code except the jump address when the routines end (ZZ in the code). The jump must be on the same page. As with any machine language routine in CHIP-8, a D4 instruction must be used to end the routine.

The code in locations ØLØØ through ØL1Ø is the initilizing sequence.

XX in location ØLØ1 is the number of pages to be transferred.

YY in location ØLØ4 is the page number at which transfer begins.

ZZ in location ØLØ7 is the high-memory location of the routine.

TAPE READ ROUTINE

ØLØØ	F8XX	AEF8	YYB6	F8ZZ	B3BC
ØLØA	F80E	A3D3	F881	BDF8	ØØA6
ØL14	F842	ACF8	ØAB9	DC33	1729
ØL1E	993A	1ADC	3B21	F809	A9A7
ØL28	9776	B729	DC89	3A28	87F6
ØL32	3335	7B97	5616	863A	212E
ØL3C	8E3A	213Ø	ZZD3	F81Ø	3D44
ØL46	3D4E	FFØ1	3A46	179D	FE35
ØL5Ø	41D4				

TAPE WRITE ROUTINE

ØLØØ	F8XX	AEF8	YYB6	F8ZZ	B3BC
ØLØA	F80E	A3D3	F8ØØ	A6F8	80B2
ØL14	F845	ACF8	40B9	92F6	DC29
ØL1E	993A	1AF8	10A7	F808	A946
ØL28	B7F8	80FE	DC86	3A31	2E97
ØL32	F6B7	DC29	893A	3117	87F6
ØL3C	DC8E	3A21	DC7A	30ZZ	~~B3F9~~
ØL46	ØA3B	40F9*	2017	7BBF	FFØ1
ØL50	3A4E	3944	7A9F	304E	D4

* Correction: ØL44 D3 F8
ØL49 F8

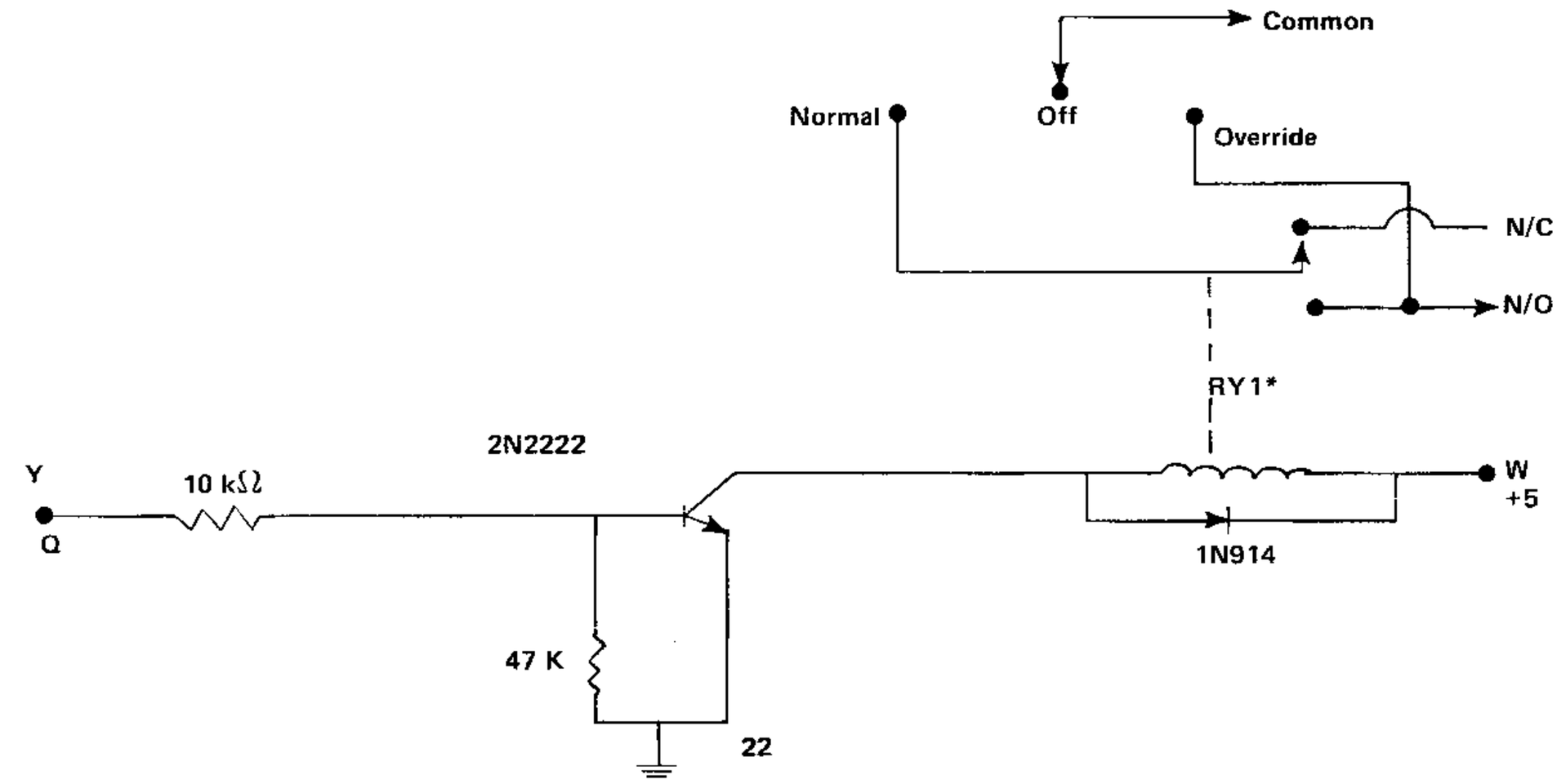
THE BENCH CONNECTION

A Low Current Relay for Switching Applications

This handy device has unlimited uses, starting with tape player controller, timer, and computer controlled switching. The circuit presented here is a tape recorder controller, although the onboard relay can control a high current type relay for turning lights off and on.

The diagram below shows the circuit to use with your VIP. A Radio Shack relay (or its equivalent) should be used. The circuit is controlled by the Q line, and can be set or reset by the machine language instructions 7B and 7A. To pull the switch down, use a SET Q, and use the inverse instruction (REQ) to release it.

The tape-write routine holds the relay down so the circuit can be used with the operating system or with the tape-write routine described in this month's software column.



*RY1 – Radio Shack 6 VDC SPDT 500 Ω 12 mA – No.275-004

GETTING STARTED WITH CHIP-8

This article is in response to requests we have received from VIP owners for more information on how to use CHIP-8.

CHIP-8 is an interpretive programming language included in the RCA COSMAC VIP. It is geared toward use in video games and manipulating graphics, but could also have applications in other areas not necessarily graphics oriented. CHIP-8 is not intended to rival languages such as BASIC or FORTRAN in alphanumeric or number-manipulation capability. However, CHIP-8 is easy to use and much more memory efficient than BASIC in appropriate applications.

An interpreter is basically a collection of machine language subroutines geared toward a specific application. A main call routine in the interpreter decodes each macro code (CHIP-8 instruction) of the program into an address, and interpreter calls the machine language routine at that address. The macro code stays unchanged in memory and is interpreted each time the instruction is executed. This

CHIP-8 IS AN INTERPRETIVE LANGUAGE

affords a savings in the amount of memory required to implement programs. Each macro code is merely an indication as to which machine language subroutine is to be executed using which data.

CHIP-8 is a hexadecimal interpretive language with an instruction set geared toward games, graphics, and keyboard sampling to manipulate a video display.

A hexadecimal interpreter provides great memory savings in that no alphanumeric character strings need to be saved; only compact codes containing the operation to be performed and the parameters.

All CHIP-8 instructions are contained in 2 bytes (or 4 hex digits). This makes changing code or the insertion of debugging instructions (i.e., a program stop) easier. This uniform size also helps to eliminate addressing errors.

The hexadecimal format and the 16 memory registers (variables) of CHIP-8 create a pseudo machine language which is very powerful for games and graphics applications. The 16 variables are used very much like machine registers but are easier to access. Most of the language's instructions refer to or manipulate one or more of these variables.

The video display is implemented by mapping the contents of the highest page (256 bytes) of on-board memory to the display screen. The video interface on the VIP generates the timing for the display including the generation of the display refresh interrupt request. The interrupt routine, which is located in the operating system ROM, controls the addressing during the memory transfer to the display screen. **The function of CHIP-8 programs, therefore, is just to arrange the data in the display page to provide the desired graphics.**

The first byte of the display page will be shown in the upper left of the display screen with successive bytes following from left to right in 32 rows of 8 bytes each. Each row of 8 bytes is actually shown on four successive scan lines to give each bit a vertical height nearly equal to its horizontal width. This vertical height is set by the display refresh routine. Writing to the display page involves specifying the position the data should occupy in the display and then transferring a list of bytes to that position.

The display page is made to look like a coordinate grid by CHIP-8. Since the display consists of a 64x32 bit format (40x20 in hex values) any bit position can be specified by a coordinate pair. The horizontal coordinates range from 00 at the left to 3F at the right of the screen. The vertical coordinates range from 00 at the top to 1F at the bottom of the screen (see Fig. 1 in VIP manual, page 15). Before transferring a pattern to the screen, a memory pointer must be set to the first byte of the pattern list. This will be explained in an example later. When the CHIP-8 display instruction transfers the pattern list to the display memory the most significant bit of the first byte in the list is posi-

IN HEXADECIMAL

FOR GAMES

AND GRAPHICS

tioned at the specified bit coordinate. This may require the pattern byte to be shifted several bits to the right and be shown in two horizontally adjacent bytes. CHIP-8 takes care of this shifting and also insures that all the bytes in the pattern list line up vertically.

There are 31 CHIP-8 instructions. They will be broken down into 8 groups: Set Variable, Branch and Call, Conditional Branch, Random Number Generator, Keyboard, Arithmetic and Logic, Tone and Timer, Memory Pointer, and Display. In the following descriptions, X and Y refer to variable names and KK is any hexadecimal constant. Use of variable Ø (VØ) is mandatory for certain instructions, and variable F (VF) is altered during the use of certain instructions.

CHIP-8 INSTRUCTIONS

Set Variable Instructions

6XKK Set the variable indicated by X to KK. Any variable may be set to any value 00 through FF by this instruction. For instance, an instruction 6304 sets V3 to a value of 04.

ASSIGNMENT

7XKK Increment the variable indicated by X by KK. Any variable may be incremented by any amount from 00 to FF. If the result is greater than FF the overflow is not saved but the value saved in variable X will be correct. Incrementing by FF appears to be a decrement by 01, FE decrements by 2, etc.

INCREMENTS

8XY0 Copy the current value of the variable indicated by Y into the variable indicated by X. The value of any variable may be copied into any other variable.

COPY

Branch and Call Instructions

1MMM Jump to the CHIP-8 instruction at memory location 0MMM. The Ms are replaced by 3 hexadecimal digits specifying the address of some CHIP-8 instruction which is to be the next instruction executed. The range of addresses possible is from 0000 to 0FFF. Addresses over 0FFF are not accessible due to the two byte CHIP-8 format. **Branching to locations under 0200 is discouraged because these locations contain the CHIP-8 interpreter code.**

GOTO OR JUMP

BMMM Add the value of variable V0 to 0MMM and jump to the CHIP-8 instruction at the resulting memory location. If the result of adding V0 to 0MMM increments the low order address past FF then the high order address is incremented. This instruction may be used to branch to one of several locations based on the results of previous events.

INDEXED ADDRESSING

2MMM Calls a CHIP-8 subroutine at memory location 0MMM. May be used to call a CHIP-8 instruction sequence from more than one place in the program and, when finished, return to the instruction immediately following the call.

CALL TO CHIP-8 SUBROUTINES

Subroutines may also be called from within a subroutine. This is known as nesting. In CHIP-8, nesting may be made up to 12 levels deep, before conflicting with the CHIP-8 memory map (see manual, page 36).

00EE Terminate a CHIP-8 subroutine. Returns execution to the CHIP-8 instruction immediately following the source of a CHIP-8 subroutine call. Every CHIP-8 subroutine must end with an 00EE instruction. Every level in subroutine nesting must end with 00EE to insure proper operation.

RETURN FROM CHIP-8 SUBROUTINE

0MMM Calls machine language subroutine at memory location 0MMM. May be used to branch execution to COSMAC 1802 machine code to perform operations not available through CHIP-8. If Control is to be returned to CHIP-8 at the instruction immediately following the source of the call, a D4 code instruction must be used.

MACHINE LANGUAGE SUBROUTINE CALL AND RETURN

CXKK Probability Branching (uses the CHIP-8 Random Number Generator)

ON . . . GOTO

Conditional Branch Instructions

3XKK Skip the next instruction if the variable indicated by X is equal to KK. Any variable may be compared to any hex value 00 through FF; if they are equal, the next CHIP-8 instruction is skipped.

4XKK Skip the next instruction if the variable indicated by X is not equal to KK. Any variable may be compared to any hex value 00 through FF; if they are not equal the next CHIP-8 instruction is skipped.

5XY0 Skip the next instruction if the variable indicated by X is equal to the variable indicated by Y. Any two variables may be compared; if they are equal the next CHIP-8 instruction is skipped.

BRANCH EQUAL

9XY0 Skip the next instruction if the variable indicated by X is not equal to the variable indicated by Y. Any two variables may be compared; if they are not equal the next CHIP-8 instruction is skipped.

BRANCH NOT EQUAL

Random Number Generator

CXKK Generate a random number, logically AND it with KK, and set the variable indicated by X equal to the result. Any variable may be set to some random value. The KK is a mask and can be used to limit the range of random numbers possible. The range can be used to provide a certain number of alter-

RANDOM NUMBERS

natives of which the resulting random number selects one. The range can also be used to set the odds in a probability branching situation. If KK = 03 there is a 25% chance of the resulting random number equalling zero. An appropriate branching structure following the random number generation completes the operation. In the following example the random number is set into variable 5. The third instruction is skipped if V5 = 00. As the range of outcomes is increased the probability of the third instruction being executed decreases.

```
C5KK
4500
1AAA  ------>
1BBB  ------>
```

KK	%A	%B	BIT MASK possibilities
01	50	50	00000001 0 or 1
03	25	75	00000011 0,1,2 or 3
07	12.5	87.5	00000111 0 to 7
0F	6.25	93.75	00001111 0 to F
FF	.392	99.608	11111111 0 to FF

Keyboard Instructions

FX0A Wait for any key to be pressed and load the hex value into the variable indicated by X. This instruction may be used to wait for an input of data. No further instruction will be executed until some key is pressed. The next instruction will not be executed until the key is released. A tone will be generated while the key is depressed.

KEYBOARD INPUT

EX9E Skip the next instruction if the variable indicated by X is equal to key pressed on hex keyboard. This may be used to check whether a specific key on the hex keyboard has been pressed. The instruction causes the low order hex digit of the variable indicated to be compared with the keyboard. If the corresponding key is pressed the instruction will detect it, and the next instruction is skipped. If the key isn't pressed, the program doesn't wait but executes the next CHIP-8 instruction immediately.

SKIP EQUAL

In the following sequence, if key A is pressed V5 will not be incremented:

```
.
.
.
630A
E39E
7501
.
.
.
```

EXA1 Skip the next instruction if the variable indicated by X is **not** equal to the key pressed on the hex keyboard. This may be used to check whether a specific key on the hex keyboard has been pressed. The instruction causes the low order hex digit of the variable to be compared with the keyboard. If the corresponding key is pressed, the instruction will detect it, and the next instruction is executed. If the key is not pressed, the next instruction is skipped.

SKIP NOT EQUAL

In the following sequence, if key A was pressed V5 would be incremented:

```
.
.
.
630A
E3A1
7501
.
.
.
```

Arithmetic and Logic Instructions

8XY1 Logically OR the contents of the variable indicated by X with the contents of the variable indicated by Y and set the variable indicated by X equal to the result. Any two variables may be logically ORed together and the result stored in VX. VF is used for temporary storage and will be altered.

OR

8XY2 Logically AND the contents of the variable indicated by X with the contents of the variable indicated by Y and set the variable indicated by X equal to the result. Any two variables may be logically ANDed together and the result stored in VX. VF will be altered.

AND

8XY4 Add the contents of the variable indicated by Y to the contents of the variable indicated by X and store the sum in the variable indicated by X. If the sum is not more than FF, VF will be set to 00. If the sum is greater than FF, VF will be set to 01. Any two variables may be added together.

ADD

8XY5 Subtract the contents of the variable indicated by Y from the contents of the variable indicated by X and store the difference in the variable indicated by X. If the variable indicated by X was not less than the variable indicated by Y, VF will be set to 01.

If the variable indicated by X was less than the variable indicated by Y, VF will be set to 00.

SUBTRACT

Tone and Timer Instructions

FX18 Load the tone timer with the contents of the variable indicated by X. The timer is decremented 60 times per second (the same rate at which the display is refreshed). This means that loading 3C into the tone timer would provide a 1 second tone ($3C_{16}$ equals 60_{10}). The maximum tone duration from a single tone instruction is a little over 4 seconds.

SET THE TONE TIMER

FX15 Load the timer with the contents of the variable indicated by X. Any variable may be used to load the timer with a hex value between 00 and FF. Once the contents of the variable are stored, the display refresh routine handles the counting of the timer down to zero. The timer stops once it reaches a value of zero.

SET THE CLOCK

Certain parts of CHIP-8 won't work if the Video Interface Chip is disabled through a machine language 61 instruction. Similarly the VIP operating system won't work if the video interface chip is disabled or defective because if the interrupts aren't executed the timer isn't decremented and the software keyboard debouncing will put the program into a non-terminating loop.

OOPS!

FX07 Load the variable indicated by X with the current value of the timer. The contents of the timer (machine register-half R8.1) may be loaded into any variable to check the current value of the timer. This is used to determine when a delay or a "time allowed" period is ended.

CHECK THE TIME

Memory Pointer Instructions

AMMM Set the memory address pointer to some memory location 0MMM. This may be used to point to any address from 0000 to 0FFF. Locations over 0FFF cannot be addressed because of the two byte format of CHIP-8 instructions. The instruction is used prior to data transfers between blocks of memory and variables or the display page.

MEMORY POINTER

FX1E Increment the memory address pointer by the value of the variable indicated by X. Any variable may be used to increment the memory pointer. This instruction may be used to vary the value of the memory pointer in accordance with some other parameter which may vary during the execution of the program.

INCREMENT MEMORY POINTER

FX29 Set the memory address pointer equal to the first byte of the 5 byte display pattern list for the least significant hex digit of the variable indicated by X. The address pointer is set to this location with the intent of displaying the value of the hex digit in the variable indicated. Only the digit derived from the 4 least significant bits of the byte is involved.

FETCH DISPLAY PATTERN

FX33 Convert the contents of the variable indicated by X to a 3 digit decimal number and store each digit in separate successive bytes beginning at the location indicated by the memory address pointer. The hexadecimal value in any variable may be converted to 3 decimal digits occupying the least significant digit of 3 different bytes and stored in memory. The use of 3 decimal digits is derived from the fact that 00_{16} through FF_{16} corresponds to 000_{10} through 255_{10}. The bytes at the memory locations containing the 3 decimal digits may be transferred to variables using an FX65 instruction. An FX29 instruction must be used for each digit contained in a variable to prepare it for display.

HEX TO DECIMAL CONVERSION

FX55 Transfer the contents of 1 to 16 variables to successive memory locations beginning with the location specified by the memory address pointer. The variables saved will always include V0 through the variable indicated by X. The memory address pointer will have been incremented by X plus 1 when the instruction is completed. This instruction and FX65 may be used to store and recall any number of variables to create multiple sets of variables.

STORE VARIABLES IN RAM

FX65 Transfer the contents of successive memory locations beginning with the location specified by the memory address pointer to 1 to 16 variables. The variables will always include V0 through the variable indicated by X. The memory address pointer will have been incremented by X plus 1 when the instruction is completed. This instruction is the reverse of the FX55 instruction.

LOAD VARIABLES FROM RAM

Display Instructions

DXYN Transfers the pattern byte list pointed to by the memory address pointer. The number of bytes to be transferred in the pattern byte list is indicated by N. The horizontal bit coordinate of the position at which the pattern is to be displayed is contained in the variable indicated by X. The vertical coordinate is contained in the variable indicated by Y. The pattern is EXCLUSIVE-ORed with the existing display in the area. If any spot in the pattern matches a spot in the existing pattern (a HIT condition) VF will be set to 01. If a pattern is shown a second time at the same coordinates it will cancel or erase itself due to the EXCLUSIVE-OR function. The memory pointer is unchanged at the completion of the instruction.

DISPLAY

00E0 The active display page is erased by setting every byte to zero.

CLEAR SCREEN

PROGRAMMING IN CHIP-8

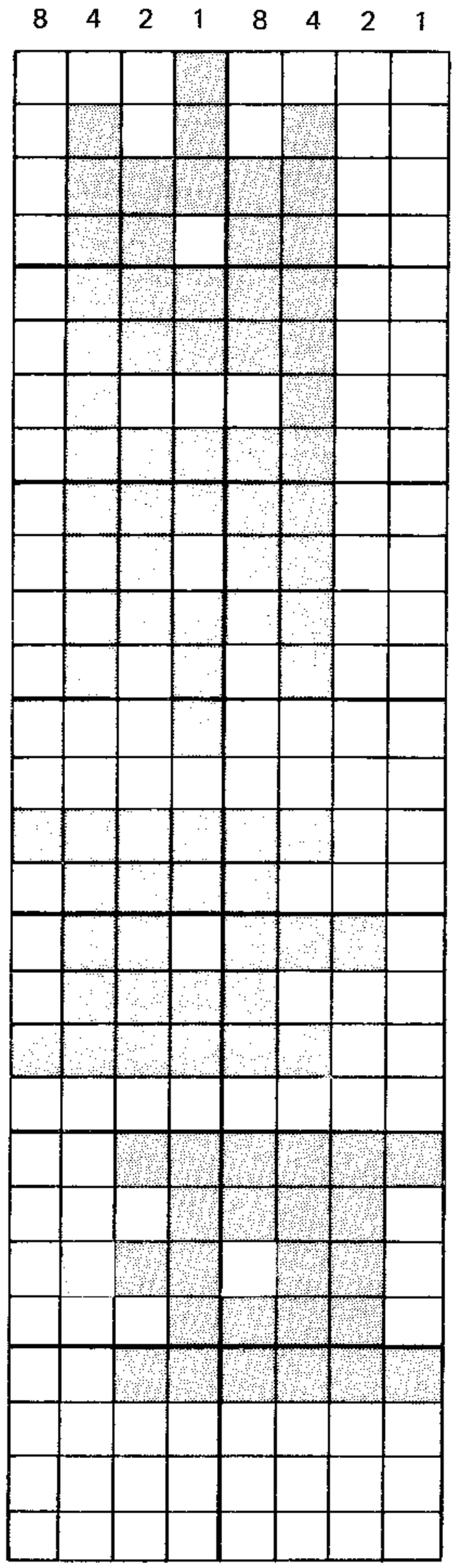
When programming in CHIP-8, as in any programming language, there are several steps involved in generating a working program. These steps include specifying the program, structuring, flow charting, coding, and debugging.

To illustrate these steps, a part of a game program will be developed. The program will be one that puts a picture of a tank on the screen and allows moving the tank in one of four directions by pressing the appropriate key. Key 2 will move the tank toward the top of the screen while key 8 moves it toward the bottom. Key 4 will move the tank left and key 6 will move it to the right. The tank will appear to face in whichever direction it is moving. To do this in CHIP-8, four different tank patterns, 1 for each direction, have to be used. In addition to specifying the operation of the program the graphics must also be generated. The bit pattern of the tank used is shown in figure 1. This must be stored along with the bit patterns for the 3 other facings.

It is sometimes possible to combine similar sections of separate bit patterns in the pattern storage area. The upper portion of one pattern must have the same bytes in a sequence as the lower portion of another pattern to do this, as shown in figure 1, which only has a limited combination of patterns. (A better example of this technique can be seen on page 37 of the VIP manual.) There is a padding of 00 bytes around the horizontal tank facings. This results in a simplification when displaying since all of the pattern byte lists are 7 bytes long and can all be shown by the same DXYN (display) instruction.

Once the program is specified it must be organized to fit into the CHIP-8 structure. This involves determining which variables are used for various parameters and how to position the program elements in memory. It is usually found to be desirable to place pattern lists and other data tables in upper memory locations. CHIP-8 subroutines and machine language code are usually located between the main routine and the patterns/tables area. So the main routine begins at 0200 and extends upward as far as necessary, calling the subroutines and data tables as needed. One purpose of this particular example is to show the power and compactness of CHIP-8, so in the program being developed, the pattern lists will be located just above the main routine. Variable utilization is simple in this case. V1 will specify the horizontal coordinate of the tank and V2 will specify the vertical coordinate. V0 will be used as a working or temporary variable. This minimal use of variables indicates that this routine could be used as a subsection of a larger, complete video game since most of the variables are still available.

The next step is to flowchart the program as shown in figure 2. The first step in the program is to initialize the horizontal and vertical coordinates of the tank. The direction in which the tank is facing is also initialized by



	ADDR.	DATA
Up	0240	10
	0241	54
	0242	7C
	0243	6C
	0244	7C
	0245	7C
Down	0246	44
	0247	7C
	0248	7C
	0249	6C
	024A	7C
	024B	54
	024C	10
Right	024D	00
	024E	FC
	024F	78
	0250	6E
	0251	78
	0252	FC
Left	0253	00
	0254	3F
	0255	1E
	0256	76
	0257	1E
	0258	3F
	0259	00
	0260	00
	0261	00

Figure 1

setting the memory address pointer to the first byte of one of the 4 tank direction pattern lists.

The second step is to show the selected tank pattern at the coordinates specified.

In the third step, keys 2, 4, 6 and 8 are checked in a loop, continuously, until one of them is pressed.

Four Direction Tank Demo Program

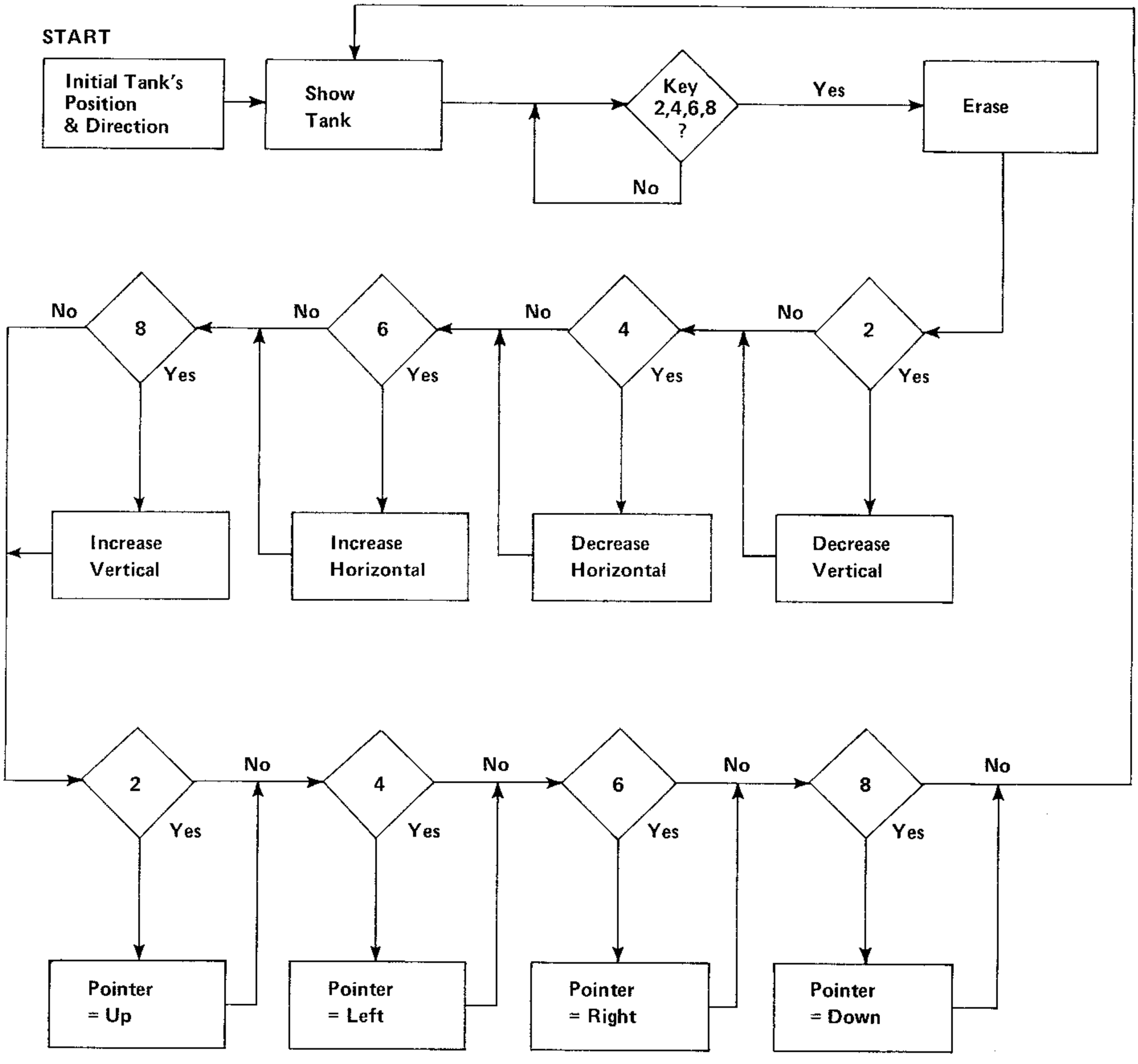


Figure 2

The fourth step is only reached after a key is pressed. At this time the tank pattern is erased. This could be done by either erasing the entire screen or displaying the tank pattern again at the same coordinates and direction. If this routine was to be part of a larger program, the erasure by EX-ORing would be preferred.

The fifth step is to alter either the horizontal or vertical coordinate depending on which key was pressed. This is accomplished by a series of tests on the value of the key pressed.

A similar series of tests will be used in step six to set the memory address pointer to the tank pattern corresponding to the direction indicated by the key pressed.

The program then jumps back to step two to show the tank at its new direction and coordinates on the display page.

Hopefully, but rarely, the last phase is coding the program in actual CHIP-8 instructions.

PROGRAMMING IN CHIP-8 (CONT'D)

Tank Movement Program

0200	INITIALIZE	6120	STEP 1	Set V1 = 20
2		6210		Set V2 = 10
4		A240		Set I = 240
6	SHOW TANK	D127	STEP 2	Show tank
8	KEY WAIT	6002	STEP 3	Wait for key: set V0 to 2
A		E0A1		If key ≠ 2 skip next
C		1216		Go to 216
E		7002		Increment V0 + 2
10		300A		Skip next if V0 = A
2		120A		Go to 20A (loop back to check next key)
4		1208		Go to 208 (loop back to recheck next keys)
6	ERASE TANK	D127	STEP 4	Erase tank
8	CHANGE X or Y	4002	STEP 5	Skip next if V0 ≠ 2
A		72FF		Move up
C		4004		Skip next if V0 ≠ 4
E		71FF		Move left
20		4006		Skip next if V0 ≠ 6
2		7101		Move right
4		4008		Skip next if V0 ≠ 8
6		7201		Move left
8	SET POINTER	4002	STEP 6	Skip next if V0 ≠ 2
A		A240		Go to 240
C		4004		Skip next if V0 ≠ 4
E		A253		Go to 253
30		4006		Skip next if V0 ≠ 6
2		A24D		Go to 240
4		4008		Skip next if V0 ≠ 8
6		A246		Go to 246
8	JUMP TO SHOW	1206	STEP 7	Start over

In the initialization step, 6120 sets V1 to 20 to specify the starting horizontal coordinate of the tank. 6210 sets V2 to the starting vertical coordinate. A240 sets the memory address pointer to the upward-facing tank pattern list and completes the initialization.

Step 2, which shows the tank pattern selected at any specified coordinates, requires only a D127 instruction. The 7 indicates that the tank pattern shown will always be a 7 byte list. The reason for the 00 bytes in the left and right tank pattern lists was to make them 7 bytes long.

The third step is more complex. With the tank being displayed at its present coordinates, the program waits for one of the four direction keys to be pressed. 6002 sets V0 to 02. E0A1 checks to see if the key corresponding to the value in V0 is pressed. If key 2 is pressed the next instruction is not skipped. This instruction is a branch instruction which jumps out of the KEY WAIT step to the fourth step. If the key corresponding to V0 is not pressed the branch instruction is skipped and the KEY WAIT loop continues with a 7002 instruction. This increments the value in V0 by two so that V0 will, in successive passes through the loop, equal 2, 4, 6 and 8. The 300A instruction checks for a 0A in V0. If V0 does not contain 0A, the 120A instruction is executed and the E0A1 instruction checks for the next KEY in succession. Once V0 is incremented up to 0A the 300A instruction causes the 120A instruction to be skipped and the 1208 instruction causes a jump to 0208 and the KEY WAIT loop starts with key 2 again. The 1216 instruction at address 020C is executed when 2, 4, 6 or 8 is found pressed and step four is executed.

The tank pattern is erased by executing a D127 instruction. Since V1, V2 and the memory address pointer have not been altered, the pattern shown is the same byte list at the same coordinates as the existing display so the tank is erased by the EXCLUSIVE OR display technique. This would leave any other patterns in a larger program undisturbed.

The next step is to change the X or Y coordinate depending on the key pressed. 4002 allows the 72FF instruction to be executed only if V0 contains 02. 72FF decrements V2 by 1 since any overflow is disregarded. Decrementing the

PROGRAMMING IN CHIP-8 (CONT'D)

vertical coordinate will have the effect of moving the tank upward since 00 is the vertical coordinate at the top of the screen. This action corresponds to key 2 since it is intended to move the tank upward. This same procedure is repeated for the other three directions and only one of the four 7XKK instructions will be executed each time this step is done.

Step six is functionally the same as step five except that the memory address pointer is affected instead of V1 or V2. In each of the parts of this step the memory pointer is set to the direction corresponding to the value of V0. Only the AMMM instruction corresponding to the key pressed will be executed.

With the new coordinates and memory pointer contents, step seven branches back to the display step to show the tank which then leads into the check for another movement instruction from the keyboard. Non-trivial programs rarely work the first time. (Errors accidentally developed in the flowcharting or coding phase or those carelessly generated when keying the code into the system will usually turn up.) It's a good idea to save all programs on tape before running them for the first time.

There are several steps in locating program errors:

- Examine the code in memory to be sure it agrees with the written code.
- Insert a debug stop at some point in the program and check the contents of the variables against what they should be. A debug stop is a 1MMM instruction that branches to itself. To stop at location 0358 put a 1358 instruction at 0358. The location of the variables is listed on page 36 of the VIP manual. Use the Memory Read function of the operating system to examine them or other stored values in your program. 1802 machine register values may also be examined as described on page 34 in the VIP manual.

Converting a hexadecimal value to a three digit decimal number and displaying it is an important application of CHIP-8 to understand. The following sequence is commented to describe how it may be done:

65C3	Set Hexadecimal value into V5
A300	Set memory pointer to A300
F533	Convert V5 to 3 decimal digits and store at 0300, 0301 and 0302
F265	Transfer Bytes at 0300 through 0302 into V0 through V2
6318	Set V3 to 18 (horizontal coordinate)
6410	Set V4 to 10 (vertical coordinate)
F029	Set memory pointer to pattern list for LSD of V0
D345	Display 5 byte pattern list at coordinates V3, V4
7306	Increment V3 by 06 to shift horizontal coordinate
F129	Set memory pointer to pattern list for LSD of V1
D345	Display 5 byte pattern list at coordinates V3, V4
7306	Increment V3 by 06 to shift horizontal coordinate
F229	Set memory pointer to pattern list for LSD of V2
D345	Display 5 byte pattern list at coordinates V3, V4

COMMENTS

The following space has been left blank for your comments. Let us know what **you** are doing with your system!