VIPER

The only comprehensive monthly newsletter devoted exclusively to

THE RCA COSMAC VIP

PRESENTS

TIPS FOR VIPS

VOLUME III

by TOM SWAN

ARESCO
P.O. Box 1142
Columbia, MD 21044
THE PAPER - VIPER - RAINBOW - SOURCE

TABLE OF CONTENTS

TO ANNE

AUTHOR'S FORWARD

Dear fellow VIPER,

The world of computers is growing -- sometimes almost too fast -- everyday. With your Cosmac VIP, you hold a precious spot in the pioneer front line of the most exciting and rewarding hobby ever to grab hold in modern times. Everyone has a computer (very nearly a true statement), and everyone owning one of these fascinating marvels has a different interest in their inner secrets. If you suspect me of being a "computer-fanatic," you are right! And I sincerely hope that the following pages will pass on at least some of the excitement that I feel while manipulating, molding, directing all those seemingly insignificant little memory bits into a program. A program that does something; whether it's a game or a "utility" routine that helps write still other programs. I cannot tell you how much I enjoy programming, and how much I want to share the experience of programming with you. But I will try, and I am thankful to you and to the fine people at ARESCO for giving me the opportunity to present my computer "discoveries" in these pages.

If you are new to computers and perhaps having some trouble getting started, my suggestion is to relax and learn as much as you are able -- a little at a time. Study the manuals that came with your VIP. They are of the best written in the market, and contain all you need for a valuable introduction to the binary world. Study other programs such as those in this book, play games, make the modifications suggested, and you will be well on your way in a hobby that I predict, once you begin, you will never put away.

To the goal of teaching computer programming in a fun and interesting way, I have made every effort to clearly mark each instruction in the following programs, what they are, what they do and why I did things the way I did. The program descriptions accompanying the listings add explanations where routines crucial to your understanding of the program are picked apart piece by piece unveiling their hidden purpose to the whole. Modifications are included to allow you to customize your game to suit your own tastes as it would be presumptuous of me to assume that my choices were the best in all cases. (And I invite my fellow authors here publicly, to do the same.) It is your computer -- you should be able to say how you want it to perform!

For those of you who have been programming for some time and have written your own programs already, the games presented here offer the unique capability of modular design, allowing experiments in higher level programming concepts without

having to pay attention to the tedious details of organizing the display, initializing variables, etc., all the things that can slow down a programming session. These programs will serve as skeletons for your own structures -- launch-pads for your own ideas -- and many possibilities to expand the programs are given, hopefully to pique your interest.

Whatever your level of programming skill, whatever your interest in computers, I hope you will "get your fingers dirty" and try some of the modifications I've suggested. There's no better way to learn, and you'll end up with a personalized program that will provide you with hours of computer enjoyment.

With this in mind, I am anxious to get started. So, good luck, and the best computer experiences to you!

Best Regards,

TOM

Tom Swan
Taxco, MEXICO 1979

P.S. --when you are ready to dig into the program listings to follow, a glance at the appendix at the end of the book will give you some valuable hints on understanding the format used in this book plus some suggestions for making use of the modifications included in the Chip-8 Interpreter supplied with your games. But I'm sure you'd like to get right to a game so think of a warm, flat desert land, somewhere in Nevada, click your heels three times, hold on tight and carefully turn the page.

Tom Swan

VIPER

July 13, 1979

V I P - O K E R

by

Tom Swan

A hushed murmur floats with the smoke in the air like a London fog poured into the back room of the casino. The heavy, buttoned leather door closes slowly pinching off the show music behind you. Music which does not belong in this room. You are noticed but not recognized, and the sound of voices muffled under the low lamps dips imperceptibly acknowledging your presence as you first survey the room with your eyes then cross to table number five. Your table. One of several green felt islands of promise set in a sea of dark browns and the reds of mystery.

Today you play poker. Thinking of the next hours, the thrill, the power, the politics of poker acts on

your heart muscle sending crescendos of excitement to the top of your throat as you take your place. To start right, that is important. The first impression is everything, and your face shows nothing of the surges you feel or the anticipating tingle of your skin. To react in any way -- except as a ploy and then with subtlety -- is blasphemy to the game. Your partners in the coming power struggle, more intricate than the dirtiest politics, stand their backs to you. Who will they be? What will it take to unlock the strategies buried in their minds hidden from all but the most perceptive detective? You are that detective. To win you must find the key to their thoughts without giving a clue to your own plots and plans. Careful now. Here they come. Their faces are obscure in the dark room though there is something about them. But, can it be!?? In spite of your best efforts, the muscles on your face yield to the shock just delivered to the usually solid foundation of your spirit. Ignoring your own desparate mental instructions to yourself, you cannot resist speaking first -- a sin unmatched in your experience.

"Terry, Rick ...," you blurt out, kicking yourself as you do so, "Tom! What are you doing here? I never thought..."

"Hi! How are you?" offers Terry Laudereau, a

cheerful smile behind her outstretched hand. Rick Simpson and Tom Swan offer their regards as they sit down facing you.

You feel the heat in your cheeks. Cheerful greetings in the poker room? Oh, most sacred of taboos broken! You will never be the same. Seeing your confusion, Terry begins again.

"You were so kind to gamble your hard-earned money on Tom's book, and we appreciate it so much," she says, "we decided to give you a chance to win some of it back!"

A thought strikes you and it melts your initial surprise. "That's very fine of you Terry," you say innocently. "I was surprised to see the three of you here at first, but since we're all sitting down, I have an idea. Let's play poker!" Two can play a hustle, you think. There is no doubt in your mind now. Terry's bluffing!

"Good idea," Terry says, "It's your deal!" and the game is on.

INSTRUCTIONS

To play VIP-OKER, first load in 14 (E) pages from tape starting with location 0000. A modified two-page display Chip-8 Interpreter is supplied with the program ready to go.

Before flipping up the run switch, please glance at the following instructions so you won't miss any of the action which begins as soon as you run the program. I know you will want to try out VIP-OKER without delay so I'll try to be brief.

First you will want to enter your own name into the computer. This must be done in ASCII encoded form using one hex byte for each letter of your name. If you don't have an ASCII chart (the 1802 machine language reference card supplied with your VIP contains one) you may easily construct one by writing down the letters of the alphabet, then starting with 41 hex for the letter A, assign sequential hex numbers to each letter ending with 5A for Z. The numbers for your name go in location 09F9-09FE. (Location 09FF must be set to 00.) If your name has more than six letters, I'm sorry to say that it won't fit in the space assigned. Maybe there's a nickname you go by that will fit. But you do not have to enter a name to play VIP-OKER. The program is preset

to use "VIPERS" in place of a name if you prefer.

Instead of competing with only one anonymous opponent as in most computer games, VIP-OKER controls the actions and strategies of three <u>different</u> players seated with you at the poker table. Terry is to your left, Rick straight ahead and yours truly, Tom, takes the chair to the right. The four of us each start with $50 and you will be the first dealer. A center arrow points to each player as the play proceeds around the table and messages keep you informed each step of the way.

As there is a lot of information on the screen, the action may seem fast at first, but after a few rounds you should be able to follow the play with no difficulty. Each player's name sits on top of the amount of cash that player has to play with. Your name (or "VIPERS") and cash are at the lower right hand side of the screen. In the center of the display, the amount of the pot is shown (if you are not a poker player, that's the total bet by all players). Your cards appear in the lower left part of the screen.

Play always begins with the computer collecting an "ante" (pronounced "auntie") of $1 from each active player. The program waits while the deck is shuffled, then your hand is automatically dealt and shown. Your

cards are displayed sorted for you from low card to high card with all pairs placed together. (This will make it easier for you to quickly read your hand.) A ten by the way is represented by the letter "T" due to space limitations on the display.

A very important number appears in the upper left corner under the word "BET." This number serves two purposes and will be constantly changing as the game goes on so you will want to watch it closely. Before any player (including you) bets, the number stands for the minimum amount the player must bet to stay in the game. When a player places a bet, this number changes to the total amount that the player is betting. That amount is added to the pot, and the arrow advances to the next player at which point the bet number again shows the minimum bet to that player.

There is a $3 limit on all bets including raises. No more than three raises may be placed during any round of betting. The raise number (which is not the amount of the raise) is shown in the upper right hand corner labeled "RSE." When this number equals 3, all players must bet the minimum or fold; no more raises will be accepted by the computer.

If the betting seems confusing, please don't worry -- poker betting is confusing unless you've had some

practice. (I speak from experience -- until setting my mind to it while writing VIP-OKER, I was a little hazy on the process.) VIP-OKER follows the correct rules for betting, however, and since the program won't let you place an illegal bet, you don't have to worry about making mistakes -- your VIP won't let you! If you do press the wrong buttons, a message will come up asking you to "Try Again."

Two nearly identical rounds of betting are normal for a round of draw poker. The first bet to be placed is called the "open," and all players are given a chance to either match the bet or raise it higher. The player to the immediate left of the dealer is given the first opportunity to open the betting. You do not have to open, and may pass by simply betting zero. Then the next player to the left is given the chance to open or pass. If all players pass on the first round of betting, the ante stays in the pot, and a new hand is dealt with a new dealer. (When it's your turn to deal, by the way, you do not have to do anything -- the computer handles the chores for you.)

To place your bet, enter the amount including any raises on the VIP's hex pad, then press Key E to "enter" your bet. To raise the minimum bet shown in the upper left hand corner, mentally add the amount you wish to

raise to the minimum and place your total bet. To fold, press Key F followed by Key E. If you make a mistake before pressing Key E, just press any key (Key 5 will work for instance) twice before hitting Key E. You will then be given another chance to bet. (The computer sees this as an illegal bet.) Remember, you are limited to $3 opens or raises and three raises are the maximum. Also you may not bet more money than you have, and just as in real poker, if you cannot bet the minimum or have no money left, you must fold. (This is a good way to force other players out of the game by the way, assuming you are trying to win. But wait a minute, what am I saying? I'm one of those other players!)

A betting round is finished -- afficionados and VIP-OKER say "called" -- when all players have either folded or contributed equal amounts of money to the pot. If you keep this one rule in mind, the whole betting process should boil down for you.

In between betting rounds, players are asked if they wish to draw from one to three cards to replace ones in their hands they do not want to keep. Again, the player to the left of the dealer goes first, and in this case the dealer always goes last. The number of cards drawn by each player is momentarily displayed to the right of the message "CARDS?" which is shown on the

screen. This number will appear directly above your name. When it is your turn to draw, you specify which of the five cards you want to discard by pressing Keys 1 through 5 on the hex pad. The cards are numbered from left to right. A small marker will appear above each card as it is selected. When you have the markers where you want them, press Key E to tell the computer you are done. The marked cards will be replaced with new cards dealt from the top of the deck, and your new hand is then resorted and shown on the screen. (Because your cards are sorted, it sometimes appears that the cards you discarded come back up on the display. This is not the case, however, and is impossible to happen. You may be dealt a king of hearts if you throw out a king of spades, for example, and the fast computer display and sorting routines can lead you to suspect the program at first when the second card is shown in the same place as the discard. Please note carefully the suits as well as the number of the cards you draw to, and the potential confusion will be avoided.)

Discarded cards do <u>not</u> go back in the deck after they are thrown out by any player and folded players are skipped over during this phase and the subsequent betting.

If you don't want any cards, you only need to press Key E when it is your turn to draw. If you try to draw

more than three cards, draw the same card twice or hit any keys other than 1, 2, 3, 4, 5 or E, you will have to start over. Any of these error conditions may be used to change your mind at any time if you make a mistake. (Just pressing Key F will allow you to start over for example.)

After all players have been given a chance to draw cards, a second betting round is begun which proceeds exactly as the first round. An important difference is if every player passes during this round, a new deal is not initiated. In that case, the program passes directly to the showdown where the winner will be determined.

The showdown normally occurs when there are at least two players active -- not folded -- at the end of the second round of betting. The main display is erased, and all four hands are displayed on the left with the player names (yours too, if you entered it!) to the right of each hand. Under the player names the rank (see chart at end of instructions) of each hand is shown. The computer indicates the winner by flashing a solid block over the rank of the highest hand. Pressing Key 0 will resume play with the arrow pointing to the winner who takes all the money in the pot. A new ante is collected, the next dealer takes charge, and your next hand will be dealt from a freshly-shuffled deck.

The three players opposing you do not play equal games and here lies the intrigue of VIP-OKER. There are four betting strategies employed, one of each assigned to each opponent at the beginning of the game. Once the strategies are assigned, they remain the same for the rest of the game. New games will have a different order of strategies for each opponent resulting in a new game every time you play!

There are:

1) One Bluff-A-Lot player
2) Two Bluff-A-Little players
3) One Bluff-Normal player

These strategies are randomly assigned, one to a player, when a new game is started. You never know, for instance, if there are two Bluff-A-Little players, or just one. And you don't know which player will be playing which strategy! The trick is to discover which players are which by forcing the bluffer out in the showdown before betting all your money away. Remember, Terry may be the Bluff-Normal player in one game, but she could be the Bluff-A-Lot in the next, so it pays (literally) to be on your toes!

Bluffing is not that obvious or easy to detect making VIP-OKER less likely to join those on your obsolete, easy-to-beat pile. A Bluff-A-Lot player will

not always bluff and may bluff a strong hand as well as a weak one. Also, super hands -- flushes, straights, etc. -- may be underrepresented by any player.

During the showdown, folded players will not be required to show their hands to the other players. Question marks -- I tell my friends they are viewing the backs of the overturned cards -- are displayed instead of the hand. If all but one player folds during any part of the betting, the last player will be paid off and the showdown will be skipped just as in real play.

The game is over when any player wins all the money ($200). As VIP-OKER duplicates the action of a real poker game, this may take one or two hours to occur. (See next section for a shorter version.) In developing the game, I have played many rounds, some to the end, some for twenty minutes or so. Even if you do not play it out to the bitter end, VIP-OKER is a fun game to play, and you may just want to set a time limit to see who has the most money at the end of a half hour. When a player has all the money, VIP-OKER indicates the winner by flashing the player's cash amount. Key 0 will automatically restart a new game at that time complete with a new order of betting strategy. No matter how many games you choose to play, you never

have to reset your computer to begin a new round.

As betting is handled independently of the main program using extensive subroutine techniques, you will be able to insert your own strategies for opponents by studying the following program details. Other variations will be discussed, too. The play will take care of itself, and you will be able to test your strategies using VIP-OKER to simulate a real game situation. (If you are just getting used to your VIP, I'll show you some simple ways to personalize your program, too. VIP-OKER is for everybody!)

Even if you do not take advantage of this unique modular feature of VIP-OKER, I hope you enjoy playing poker with "us." Have fun!

POKER HANDS

Poker hands are ranked as follows, the first in the list being the highest possible hand. Ties are broken by the highest card held in the hand. In the very unlikely event that two hands are identical, the player with the highest player number will be the winner. Terry is player number 1, and you are player number 4. This differs some from standard poker, but is a situation that will rarely if ever occur for you.

(It has never occurred for me!) Besides if you are involved, you will win the hand as you have the highest player number.

1) Straight Flush -- five cards of the same suit and in sequence. The highest possible poker hand is the Royal Flush, a straight flush with the A, K, Q, J and 10 of one suit. VIP-OKER's name for this hand is "STRTFL."
2) Four of a Kind -- four cards of the same type, for example, four 10's or four 2's. The fifth card does not matter. VIP-OKER calls this hand "FOURS."
3) Full House -- two cards of one kind and three cards of another. K-K-K-4-4 is a full house. In the event of two full houses, the player with the highest 3-of-a-kind wins. (K-K-K-4-4 beats 6-6-6-A-A for example.) VIP-OKER names a full house, "FULHSE."
4) Flush -- all five cards of the same suit. Having all hearts, but not in sequential order, would give you a flush. VIP-OKER calls a flush a "FLUSH" which is akin, I suppose, to calling a spade a spade.
5) Straight -- all five cards in sequential order but not of the same suit. 4-5-6-7-8 is a straight. Ace is always high after a king. Therefore an A-2-3-4-5 would not be a proper straight. VIP-OKER uses the

word "STRT" for straights.

6) Three-of-a-Kind -- three cards of the same type. 4-4-4-6-K is a hand with three-of-a-kind. VIP-OKER will tell you you have "THREES."

7) Two Pair -- two cards of one kind and two of another. 7-7-Q-Q-K is a two pair hand. The player with the highest pair will win, but if that still results in a tie, the fifth card determines the winner. (7-7-Q-Q-K beats 7-7-Q-Q-8 for example) VIP-OKER calls this hand "2 PAIR."

8) One Pair -- two cards of one kind as in 6-7-9-A-A. Ties are broken by the highest of the three remaining cards. VIP-OKER calls this hand "1 PAIR."

9) Zilch -- none of the above. All cards out of order, different and of unlike suits. Ties broken by highest card in hand. 2-4-7-9-A beats a 3-6-7-Q-K for instance. VIP-OKER will spell out "ZILCH" for these hands.

As VIP-OKER automatically finds the correct winner, you don't have to worry about accidentally calling a full house "THREES" a common error among beginners. VIP-OKER will help you learn the above (which may come in handy next time you're in Vegas), but you might want to keep this chart nearby while you play unless it is already familiar to you.

ABOUT THE PROGRAM...

VIP-OKER turned out to be a big program. It fills practically every nook and cranny in my VIP's 4K of memory, and five rewrites while designing the game were needed to squeeze enough room out of my already bulging memory chips to continue programming! Extensive use of machine language subroutines helped save some space, but only where I was sure the comparable routines in Chip-8 would have taken more room. (Many times a routine will be shorter than its machine language cousin by virtue of an interpretive language such as Chip-8, but not always.)

While the program listing supplied with your game explains each instruction in detail, some additional explanations are necessary for those of you who want to dig in and discover how things work. Many of the routines used by VIP-OKER may be adapted to your own game programming. In particular the Message center sub and new number display may make nice additions to your subroutine library. I'll concentrate on those and other features that will allow you to customize a version to your own taste.

For the following comments, please refer to the program listing for VIP-OKER which follows this chapter.

Immediately following the Variable Assignment is a list of all subroutines and data blocks. This may be used as a table of contents to the program to find the routine being discussed as my comments will not necessarily follow in the same order as the listing. (Some parts need to be understood before preceding to others, so please excuse the apparent disorder -- I've tried to order things for ease of understanding, though some sections found a home in memory for the simple reason that there just was no place else for them to go!)

SIMPLE VARIATIONS TO TRY

VIP-OKER was designed to play a tournament style game of a length approaching a real life poker situation. This takes at least an hour which may be too long for you, or maybe you just want to quickly demonstrate your computer to a friend. The easiest way to adjust the length of the game is to give each player a lower or higher amount of money to start with.

The instruction at 0308 at the game beginning sets the value of all players' cash. V0 is set equal to the hex number equivalent of the amount of money you want to program, with a limit of 3F hex or $63

decimal for a truly afternoon-gulping round of poker. I've found that a game of $25 (19 hex) makes a good compromise for a medium length game, but even lower amounts may be programmed. Less than $10, however, would probably limit the game's action to a few rounds of feverish folding, although it may make a nice "demo."

In addition to the change at 0308, you must enter a new winning amount at location 050A in the Payoff Sub so VIP-OKER knows when to stop the game when one player wins all the money. The instruction there causes a skip if V0 is <u>not</u> equal to the total amount of the four players' cash in hex. This is preset to C8 hex which is equal to $200 decimal. Whatever value you place in location 0308 must be multiplied times four and set in the instruction at 050A. If you entered 19 hex for a $25 game, then 64 hex for $25 x 4 = $100 decimal goes in the instruction at 050A which would be changed to 40<u>64</u>;SK≠64 -- Skip if V0≠64=$100.

* * *

In VIP-OKER's instructions, I've showed you how to enter your own name which will then be shown on your display. If your name happens to be Terry, Rick or Tom (not the most uncommon coincidence I could think of) you may want to enter new names for your opponents to avoid the obvious confusion. Or perhaps you would

like a version with the names of your family. Or play against your best friends or trounce your worst enemies.

This change is as easy to make as entering your own name. Working with ASCII codes for the letters of whatever names you want, enter name #1 at 09E4; name #2 at 09EB; and name #3 at 09F2. In each case you have six spaces for each name. If you want to insert cross hatch marks as in the version supplied on tape, these may be programmed by inserting ASCII 40's. Spaces are ASCII 20's. The seventh byte in the string, or the byte immediately following the name must be a 00 null character. The program prints characters until it reaches a 00 null. *

Mickey, Donald and Goofy would fit, for instance, as no name is more than six characters long. Most punctuation is available in the character set and all the numbers, but some symbols were replaced by subroutines not having anywhere else to go. Experimenting will tell you which symbols are no longer there, though if you have a copy of the Character Designer program from Pips for Vips, the character set may be examined in full more easily. (It is located at 0C00-0DFF and will have to be relocated to 0600-07FF to view with the character designer operating in mode two.)

* * *

* Please see instructions for CHIP-8 Messager in PIPS For VIPs I by the author for further instructions on displaying text in CHIP-8. (PIPS I available from ARESCO)

The betting limits may not be easily changed as they are tied in to several sections of the program. The listing is detailed to the point where such a change could be constructed with careful study, but would require a total rewrite of the betting module. If you plan to write your own module, as suggested later, then perhaps you will want to consider this change at that time.

* * *

A very simple way to change the bluffing strategies exists, and you may want to play around with some variations. At 0C0C, four bytes are stored representing the four possible bluffing strategies. The 7F byte is the Bluff-Normal player, the two 0F's the Bluff-A-Little players, and the 03 the Bluff-A-Lot player. Any four bytes may be inserted here with the general understanding that the higher the number, the less the player assigned with that number will tend to bluff. An FF byte will result in a player who may not bluff at all (actually only 1% of the time) while a 00 byte will assure bluffing on every round for the player assigned this strategy.

These four bytes are inserted into a CXKK random number instruction to form the basis for a player's decision to bluff or not to bluff. The bytes you insert at 0C0C will be used as KK masks for the C0KK allowing

fast programming of new betting strategies. After a game is played, you may examine the bytes to see who was which player. Terry's strategy is at 0C0D, Rick's at 0C0E and Tom's at 0C0F. The bytes will not necessarily be in the same order you programmed them, a process described in the next section. When you play, you still won't know who has which of the new strategies you inserted.

This completes the ideas for simple variations. Next I will cover how some of the subroutines operate, and give suggestions for further modifications that may be made for custom games.

PROGRAM DESCRIPTION

One of the more interesting sections of VIP-OKER is the routine that sets up the bluffing strategies at the beginning of each game. The first subroutine involved is at 097E which is called from location 0300 before the display comes on for the first time and at the start of new games. This sub shuffles the bluffing masks at 0C0C which were just discussed. The shuffling uses two random numbers to index two of the four bytes @ 0C0C which then have their order reversed by a machine language subroutine (MLS) at 0B00. This is done 16 times to assure a good mixup. The order, then, of the bluffing masks determines which player will use which mask to decide when to bluff on its next bet. This order does not change for the rest of the game.

Though the three opponents in VIP-OKER each play their own game with their own bluffing strategy, their own cards and betting, all three actually operate from the identical subroutines by the use of a single variable, V4, to tell the routine which player is betting, drawing cards, folding, etc. For this reason, you will seldom see a routine or a section devoted strictly to one player, a fact which is responsible for squeezing this program into the VIP's 4K of memory.

V4 is called an index, and its use if demonstrated by the Bluffer routine which begins at 03CE.

At this point in the program, an opponent has already evaluated its hand producing a number in V2 which will be used to determine whether to fold, stay (bet the minimum) or raise. This number, or weight, is constructed with the routine from 038A to 03CC.

At 03CE, the player's bluffing flag is checked to see if that player is already bluffing as this may not be the first time that the routine is entered. (Once any player begins to bluff, the player will continue to bluff making its play consistent.) These flags are at 03FD-03FF and the proper flag for any one player is obtained by setting the "I" memory pointer to 03FC (which is one byte less than the position of the first flag at 03FD) and adding V4 to "I" in order to address that player's flag. Since V4 is always equal to the number of the player betting at this point, only one of the three flags will be tested each time this routine is executed. And that flag will always correspond only to the player currently involved, meaning that the routine will operate independently and consistently for each player. The use of V4 as an index to control the three opponents betting, etc.,

is extensive in VIP-OKER, and would be a good feature to include in other games that operate with more than one opponent.

Provided the player is not already bluffing (its flag is equal to 00), the Bluffer routine begins the process of deciding whether to begin a bluff. This can happen at any time during play, by the way, and not only during the opening bet.

At 03D8-03DC the same V4 index technique is used to obtain a bluffing mask from the four bytes which were shuffled at 0C0C. Thus the same byte is always obtained for the same player each time this section is executed.

The instruction at 03DC lets V0 equal the byte addressed by "I" indexed by V4. (Indexed means the value of V4 is added to "I"). At 03DE, "I" is set to the address location 03E3 which holds the second half of the Chip-8 instruction C0KK at 03E2. The value of V0 is placed in this instruction to form a different C0KK instruction corresponding to each player's bluffing mask at 0C0C. As this is done before the C0KK is executed at line 03E2, the routine will operate independently for each of the three opponents.

When the CoKK is encountered, depending on the value of the KK bluffing mask, the value 00 will be

generated into V0 a certain percentage of the time. With a low number mask, this will occur at a greater frequency, and the routine will cause a player to bluff at this frequency. A high number mask will prevent a player from bluffing more than a few times during a game.

A section of a program that creates instructions before they are used is called "self-modifying code," and most literature will recommend staying away from writing such code. In most cases you should avoid the technique for two reasons. One: it prohibits the program from being placed in a PROM where the memory bytes are fixed and cannot change. Two: it tends to be the most difficult and sometimes exasperating code to debug.

But when approached with caution and care, self-modifying code is something to keep in mind for your game programming. In a situation such as the Bluffer routine for VIP-OKER, it turns out to be the simplest way to do the job -- the first requirement for the use of such a technique. It is a programming taboo to be broken with the greatest respect.

The actual bluff is accomplished by adding a constant value to the weight of the hand which was discussed earlier. This constant is at location 03E8

where the instruction 7206 adds 06 to the value in V2. You may experiment with higher or lower constants which will affect the subtlety or obviousness of the bluff. The present setting seems to produce a good range of $1, $2 and $3 bluffs, but if you make changes to the betting module, you may have to adjust the bluff constant to reflect those changes.

To eliminate bluffing altogether, enter 1DBA in place of the instruction at 03CE. Actually this change will make VIP-OKER more competitive as all three opponents will tend to play a conservative game though you may be able to bluff your way through to a win. This change must be made when experimenting with new betting modules in order to evaluate their effectiveness. (In fact, this is how I fine-tuned the module supplied with your game.)

We have been discussing the "betting module" up to now without really explaining what this is, and how it works. The module is actually broken up somewhat due to the size of VIP-OKER and the difficulty I had finding room for this routine. But, each section performs its own function so we can talk about them separately though I will continue to refer to the whole as the "betting module."

Things begin way back at 0C90 where the betting

module, a subroutine, first takes a look at the V4 Index to decide if one of the three opponents is betting or if it is the user's turn. If it is one of the opponent's turns, a jump from line 0C92 to 038A is performed. If it is your turn to bet, the routine passes on to lines 0C94 to 0CB2 in which your keypress is accepted, tested for legality or if you are folding and jumping back to 0C8C to print an error message if needed. You may eliminate yourself from the game entirely and construct a routine that would permit a fourth computer player to join the game. Such a module would need to return a legal bet, raise, pass or fold, the mechanics of which do not need to be studied in order to write a module that works. To fold <u>any</u> player (the fold routine uses the V4 index to "know" which player is folding) simply perform a jump <u>from</u> your new subroutine to 05B4 with a 15B4 instruction. (See line 0CA4 for an example.) The player's name will be replaced with the word "FOLDS" and all other processes that need to be done are automatic. To bet, simply place the total amount bet in V1 and perform a jump to 05E4 with a 15E4 instruction. All betting subroutines jump to either 05B4 to fold or 05E4 to bet and will return with the 00EE instruction at line 05EA which marks the end of the module.

I purposely kept the input/output of the betting module simple to encourage experiments with different strategies. As the computer actually sees your own play as just another module, replacing yourself with your own computer strategy is simplified.

To demonstrate this, you may let the computer bet for you by making the following changes:

```
0C90 MOD     :138A   BET --Go betting modules all
                         players
03CE BLUFF   :A3FB ;BLFLG--Set "I" to flags less one
03D8          AC0B ;MASKS--Set "I" to bluff masks
                         less one
03EC          A3FB ;BLFLG--Set "I" to flags less one
```

Also make the following patch:

```
03B4          15EC ;PATCH--Jump to patch at 05EC

05EC          4404 ;SK≠04--Skip next if ≠ player #4
  EE          ABC2 ;EVAL4--Set "I" to player #4
                         evaluation
  D0          F165 ;GET  --V0:V1 ←M(I) (patched instruction)
  D2          13B6 ;RET  --Return from patch by jumping
                         to 03B6
```

The problem with this change is that you will also be assigned a bluffing strategy, but you won't know which is yours! For this reason it is best when fine-tuning a betting module this way to eliminate the bluffing as described earlier. As you can see how the computer bets a hand, this gives you a valuable insight into the working of the betting module. You will still

have to draw your own cards and restart the computer after a showdown by pressing Key 0, but I'll show you how to arrive at a pure computer game in a moment.

One other possibility in the Bluffer routine is to eliminate the shuffling of the Bluffing masks. If you do this, the order of masks at 0C0C will now be fixed so you may predetermine which player will use which mask and test the effectiveness of various new values. If you make the changes to allow the computer to bet for you, Player #1 mask will now be at 0C0C and Player #4 (your) mask is at 0C0F. Eliminate the shuffling of the Bluffing masks by entering a 1302 instruction in place of the subroutine call at 0300.

The Draw cards controller sub at 084C controls all players' drawing of cards again using V4 as an indicator to know which player is drawing. Most of this subroutine is devoted to controlling the display, calling the message routine and advancing the arrow in the center of the "table." You may easily allow the computer to draw cards for you by making the following change:

```
086A          186C  :NOP -- Jump to next line
```

Now you have an automatic VIP-OKER which both bets and draws cards for all four players! One small problem

is that the computer won't tell you which cards it is drawing for you though it will say how many cards it drew. As a suggestion, the routine called BLINK at 0564 could be replaced by a sub of your own design to indicate which cards will be discarded using the information (discussed later) stored in the evaluation for your hand at 0BC2. (The name BLINK is a misnomer, by the way, and describes an earlier idea I had to blink each card drawn. This was confusing to use, however, and I changed my approach. But I was already used to the BLINK title so it stayed.)

If you truly wish to automize your program, you will have to make one more change. In the WINER (winner) sub at 06D8, the hands are further evaluated by a process of elimination (the winner simply pops out of the routine like a bar of slippery soap from your hand in the shower) and the flashing block indicates who had the highest cards. You must press Key 0 to restart amother round, but an auto-restart is easy to achieve. For this, make the next changes:

```
0782      60FF ;V0=FF --V0 passes value to timer sub
0784      2620 ;TIMER --Do sub--wait before restarting
0786      00EE ;RET   --Return (auto-restart)
```

The showdown phase of VIP-OKER will now wait about

four seconds until automatically restarting a new round, and though the win block will not now flash, it will still indicate the winner if you happen to be watching. The value set into V0 at 0782 may be lowered to allow a faster restart if you want.

Your VIP-OKER will now play automatically to the end of the game. This version may be stored on tape and used as an "attract mode" for times when you only want your computer to operate as background. If you are really serious about writing and testing better modules, perhaps you will want to eliminate all the timing loops by carefully going through the program listing and inserting jumps to pass all the TIMER sub calls. This would result in a high speed poker game which could be used as a bench test for various betting strategies though the display will act at fast motion like an old time movie!

While the weighing of a hand is an important part of the betting strategy (see lines 038A-03CC) the actual bet is determined with a routine at 0DBA-0DFC, which is still part of the "betting module." The listing's comments are self-explaining. What happens here is a fairly simple series of tests to set a bet in V1 based on the weight value in V2 returned from 038A-03CC.

The thresholds may be adjusted to fine-tune the betting or you may write your own routine in place of

the one here. Please note the two instructions at lines 0DBA and 0DBC which add a constant to the weight before any raises are made to encourage opening. Without this, players seem to bet much too conservatively. Also note line 0DC8 which prohibits a player from folding if there were no openers yet indicated by the betting minimum in V9 equal to 00.

Some deficiencies in the betting will become obvious after you play and examine the game for a while, but VIP-OKER was designed with expansion in mind, and the information exists in memory to vastly improve play. Some ideas are listed and numbered here to suggest advanced improvements that could be made to the betting module.

1) The opponents do not bluff on the number of cards that they draw. In the section to follow, I will discuss how a player's hand is evaluated, and you may easily cause a player (by rewriting the betting module) to bluff the number of cards it takes during the draw.

2) The opponents do not know how many cards the other players have taken, and do not take this information into account in their betting and bluffing strategy. This would be a highly desirable improvement to make to the game, and foreseeing this, I have caused the draw cards subroutine to store at 0C00-0C03 how many cards are drawn by each player indexed as usual by the V4 index. Cards

drawn by player #1 are at 0C00, player #4 at 0C03, and a revised betting module may use this information to adjust its betting strategy.

3) The opponents do not examine other hands during the showdown to check if a player was bluffing. A modification including this feature would cause the players to adjust their play to flush out bluffers, and bet them into poverty! The information for determining which player has what type of hand will be discussed next.

Players' hands are dealt from the deck of cards stored at 0BCC to 0BFF. The fifty two cards are each represented by one byte split into two, four-bit nybbles with the first four bits representing the suit and the last four representing the card type or number from two to ace.

The "cards" in this deck always remain in the memory space at 0BCC-0BFF. However, their order is changed by the Shuffle sub at 0600. The Chip-8 portion of this subroutine causes V0 and V1 to be set to random numbers from 00-33 hex (33=51 decimal) by two calls to the GENER sub located at 0614. These two variables are used by the MLS at 0B00 to index two registers into the deck of cards. The bytes at these two locations are then switched, each going where the other was. This is repeated 255 times.

By experimenting, I determined the minimum shuffle

to require about five seconds for a good mix. The Shuffle routine, which lasts about four seconds, is called twice for an eight second total shuffle of 510 exchanges during the opening rounds of play, and serves a dual purpose of providing timing between the messages "DEALER" and "SHUFFLING." (See lines 031A-0326)

Each hand is dealt into its own five-byte area, one area for each player, beginning at 0B90 as marked in the program listing. The subroutine that deals the hands (named "HANDS") is at 062A. In a real card game, impure shuffles are checked by dealing one card to each player around in a circle thus avoiding the problem of the same two cards appearing together in a player's hand even if they happen to remain together in the deck in spite of shuffling. VIP-OKER deals all five cards of a hand at once to each player -- an easier method to program -- and requires a more thoroughly shuffled deck than in a real game. This produces the same results, however, provided the deck is very well shuffled.

If you notice that the same cards tend to appear in the four hands during showdowns, this is only normal as a maximum of 32 cards, well over half the deck, may be used during play. It is reasonable to assume that half of the cards in one half of the deck will still be there following a shuffle no matter how well mixed up the deck

becomes. But we are leaving computer theory and fast approaching poker theory. Let's return to the binary world.

The hands, after being dealt, are evaluated by a call to the Evaluation sub located at 0648. Actually this subroutine only serves to call two machine language subs that do the work of sorting and evaluating each hand. The sorting sub at 0B16-0B17 is responsible for placing the cards in order, needed for displaying your hand and to make determining pairs and straights easier. (Any sorting routine automatically brings like elements together.) This sorting routine works quite well for sets no larger than a dozen or so bytes, and takes up very little space, but it is quite inefficient when used on large amounts of data. It may be used in your own programs by changing the byte at location 0B1F to the number of bytes for sorting minus one, setting the "I" pointer to the first byte of the set, and calling the routine with an OMMM instruction where MMM= the address of the subroutine. Three branches must be changed in the routine if you relocate it, however. These are at 0B21, 0B30 and 0B37.

The hand evaluation machine language routine at 0A00 is the program's most lengthy section. It may seem complex, but really isn't. All that happens is a series

of tests to check for pairs, threes-and fours-of-a-kind, straights, flushes, etc., ending with a value in RE.1 that represents one of the nine possible hands. A look up table at 0AEC contains matching values to the numbers in RE.1. These values represent the correct order for the poker hands, and one value will be stored with each hand in the first byte immediately following the hand. (These are labeled EVAL1, EVAL2, etc., in the program listing, and head a 10-byte area for each hand.)

The MLS evaluation also recommends discards and provides additional information about each hand, information which may be used by new betting modules as suggested earlier. Each player's evaluation area, following that player's hand, is constructed with the first byte representing the hand type ranking from 0-8. (See, for example, location 0B95.)

If there are any pairs in the hand, the bytes immediately following the first one in the 10-byte evaluation area indicate how many of which cards are in the hand. A byte 28 would mean there are two eights in the hand while 3A would stand for three tens. (Remember the hex letters! A=10, B=Jack, C=Queen, D=King and E=Ace.) If there are two pairs, the two bytes following the evaluation byte indicate the pair information. The lowest value pair will always be first as the hand was

sorted before the pairs information, if any, was created. Following any pair information will be an FF stop byte to indicate that the information is completed. If there are no pairs in the hand, the FF stop byte will immediately follow the evaluation byte in the first location of the 10-byte area.

After the FF stop byte, the evaluation routine deposits the cards that it recommends a player throw away during the draw part of play. The cards listed here are in the same format as described for the deck of cards at 0BCC. The recommended discards are followed by a second FF byte to indicate the end of the evaluation section. If there are no discards recommended, the FF byte immediately follows the other FF byte discussed above.

For example the evaluation:

02 26 28 FF 14 FF -- would stand for a rank of 02 (the first byte) indicating a two-pair hand. There are two sixes and two eights in the hand (the second and third bytes), and the FF byte after the 28 indicates that this is the end of the pair information. Following this byte, the 14 byte means that the computer recommends discarding the four of hearts (the 1=hearts; see location 0BCC for the other suits), and the next FF byte signals the end of the evaluation section for that hand.

06 24 39 FF FF -- is read: "you have a full house (06) of which there are two fours (24) and three nines (39) and I do not recommend throwing away any cards."

In the event of a zilch hand, by the way, the computer recommends throwing away all of the cards, a good but unfortunately illegal (to poker) suggestion. The draw routine watches to be sure a player only draws the first three cards and no more. These cards will be the lowest in the hand as the cards were pre-sorted from low to high before being evaluated, a good strategy for the draw.

If you want to cause a player to bluff by taking only one card and betting accordingly, you only need to write a routine that inserts an FF stop byte in the proper place of the evaluation section. The FF byte must follow the cards you want to discard and the rest is automatic. However, you will now be faced with a situation where all players (except you of course) are able to bluff in this way, but not able to react to a like bluff by another player. To add this feature, you will also need to make use of the cards drawn information stored at 0C00-0C03. This was discussed earlier.

Your hand is sorted and evaluated along with the others, a fact which makes programming a pure computer match easy as detailed before. The other players do

not look at this information (after all, that would be cheating) but perhaps during the showdown, your own design of betting module could look at each active hand to decide if that player could have been bluffing. A table could be built up with this information to allow every player to decide which player bluffs a lot and which only a little.

When a player folds, all of his cards are set to FF's thus obliterating that hand from memory. During the final call to the evaluation sub just before the showdown, all FF's in a hand result in an evaluation of FF in the first byte of the player's 10-byte evaluation area. This also causes the backs of the cards to be displayed (all ???'s).

Another improvement I can think of is to encourage drawing to straights and flushes provided the player is rich enough to attempt such a maneuver. This would require a further look at the player's hand as the evaluation sub does not now indicate near misses.

The sub that controls the arrow is a simple little thing, but deserves a mention as you may wish to use it for other games. Please turn to location 0400 in the program listing. At 0405, a byte is retrieved from location 0420 (labeled AINDX for Arrow Index) and placed in V0 by the F065 instruction at 0406. This byte

represents the last known position of the arrow (generated the last time this sub was called) and is first used to erase the current arrow on display. The display of arrows is controlled by a subroutine at 0422 which first sets "I" to the proper bit pattern depending on the value in V0 and then displaying (or erasing) the pattern with the DCD8 instruction at 0430. On the first time through this routine (when there is no previous arrow to erase) VF is tested to see if it is 01 in which case the information displayed touched another display (the previous arrow) pattern. If VF=00 though, then no arrow was there previously and another call to the sub at 0422 resets the arrow just displayed. This makes initializing the display easy and avoids flags and things to know when the routine is initializing and when it is not.

The arrow index indicates which player the arrow points to on return from the routine. This number is in V0 at that time, and will be set into V4 by the betting and draw subs that called the arrow routine. V4, remember, is subsequently used to index certain flags and information to independently control the action of individual opponents and strategies.

The Message Center sub at 083E controls the printing of messages in the display to ask you to "PLEASE BET"

and to tell you who is the "DEALER" and when the program is "SHUFFLING" etc. The same routine displays all the messages and represents a departure from the normal VIP way of doing things. In other words by figuring out the bit patterns of letters on graph paper and then displaying the bits with a DXYN instruction.

Every time the Message Center is called, several things happen. First the old message is erased by a machine language subroutine at 093A which sets the message window to all zeros. This was easier to program than writing a routine that erases messages by redisplaying the old one on top and then proceding with the new message.

Any routine calling the Message Center first sets V0 equal to a number representing whatever message is to be displayed. The machine language subroutine at 094A uses the value of V0 to set the "I" pointer (which is in reality register RA to the computer) to the address of the proper ASCII string for that message. VC and V0 are then set to the XY coordinates of the display's message window, and the PRINT part of the Message Center sub at 0846 calls the Messager sub located in the Chip-8 Interpreter supplied with your game. The Messager sub requires the use of the DCD5 instruction at line 0848 and will print any ASCII string addressed by "I" up to 16 characters long. All strings must end with a 00 null

character or Messager won't know when to stop printing. You may use this in your own game programming, but along with the modified interpreter, you will also need to include the character set from location 0C00-0DFF.

If you would like more information on programming poker for computers, an excellent article appeared in the July 1978 issue of Scientific American which should be available in a library near you. The article features a facinating color display for the game plus giving insights into various betting strategies you may want to adapt to VIP-OKER. For more detailed discussions on poker probabilities and strategy, most bookstores carry a selection of inexpensive books on how to play and bet a poker hand. I'm sure fellow VIPers would like to know of your improvements as you make them, and I know I'll be scanning the pages of VIPER for your comments.

A poker fanatic visitor happened to come to my apartment here in Mexico the other day while I was testing out a variation of VIP-OKER. I mentioned that an advantage of playing against a computer is the freedom you have to express your joy at drawing to an inside straight or a full house. Jump up and down if you want -- the computer won't know. His comment was, "Yes, but you don't have the satisfaction of pulling your glasses down and peering with red hot daggers into

the very heart of your opponent causing the sweat to break out on his brow by your telling gaze."

True. Then I started thinking. That computer was betting pretty well against my last couple of straights and threes. But, no, it couldn't be possible. It did beat me though.

Now I know it's just a bunch of wires and transistors and things, and the names of the players are just display patterns that aren't really even there. Still you never can be sure, so I've been practicing my best poker face ever since. Just in case. Have fun.

VARIABLE ASSIGNMENT

V0 - Various--passes values to subs
V1 - " " "
V2 - " " "
V3 - " " "
V4 - Player index (number 1-4) in betting and other subs
V5 - Total bet player #1 (on a given betting round)
V6 - " " #2 " "
V7 - " " #3 " "
V8 - " " #4 " "
V9 - BET$ - minimum bet or amount being bet
VA - RSE$ - raise number (number raises during betting)
VB - POT$ - total bet all players + ante
VC - X coordinate all display information
VD - Y " " "
VE - Index to deck of cards (finds top of deck at all times)
VF - Flags, immediate uses, etc.

MEMORY MAP

0000-02FF -- Chip-8 Interpreter - two-page display
0300-0BFF -- Program subs, data
0C00-0DFF -- Character set (with sub routines interspersed)
0E00-0EFF -- 2 pages for display refresh

ROUTINES AND DATA

0300-0388 --	Main loop -- controls game	(BEGIN)
038A-03CC --	Part of playing betting module	(BET)
03CE-03F2 --	Bluff decider part of betting module	(BLUFF)
03F4-03FE --	Reset bluff flags sub	(ZERO)
03FC-03FE --	Bluff flags	
0400-0432 --	Arrow sub	(ARROW)
0434-0450 --	Dealer sub	(DEALR)
0452-04F2 --	Bettor sub -- main controller	(BETTR)
04F4-0516 --	Payoff sub	(PAYOF)
0518-0522 --	Amount sub	(AMT)
0524-0528 --	Cash index sub	(CSHIN)
052A-053A --	Dollar sub	(DOLAR)
053C-0548 --	Ante sub	(ANTE)

```
054A-0562 -- Field sub                                    (FIELD)
0564-05A2 -- Blink sub (entry @ 0568)                     (BLINK)
05A4-05B2 -- Draw sub                                     (DRAW)
05B4-05EA -- Fold sub                                     (FOLD)
0600-0612 -- Shuffle sub                                  (SHUFF)
0614-061E -- Generate RND # sub                           (GENER)
0620-0628 -- Timer sub                                    (TIMER)
062A-0646 -- Hands sub                                    (HANDS)
0648-065E -- Evaluation controller sub                    (EVAL)
0660-0668 -- Cards sub                                    (CARDS)
066A-0674 -- Blank sub                                    (BLANK)
0676-0690 -- Show five sub                                (SHO-5)
0692-06A6 -- Decode sub                                   (DECOD)
06A8-06B8 -- Show card sub                                (SHOCD)
06BA-06CE -- Rank sub                                     (RANK)
06D0-06D6 -- Set "I" sub                                  (SET I)
06D8-0796 -- Winner sub                                   (WINER)
0798-07B4 -- Numbers sub (entry @ 079C)                   (NUMBS)
07B6-07DC -- Name printing subs  (N= 1-4)                 (NAMEN)
07DE-080A -- Money printing subs (N= 1-4)                 (MONYN)
080C-081E -- Bet Raise Pot sub                            (BRP)
0820-083C -- BRP value subs              (BET$)(RSE$)(POT$)
083E-084A -- Message center/Print subs       (MCENT)(PRINT)
084C-089C -- Draw cards controller sub                    (DRACD)
089E-08B4 -- Legal bet test sub                           (LEGAL)
08B6-08BE -- Store V0 @ I sub                             (STR4)
08C0-08FF -- ASCII strings -- hand types

0900-0925 -- MLS--high sub
0926-0934 -- Pick up pairs sub                            (PICK)
093A-0949 -- MLS--clear text line
094A-0958 -- MLS--look up message
0959-0966 -- MLS--look up hand type  (sets "I")
0968-097C -- Count active players sub                     (COUNT)
097E-098E -- Shuffle bluff masks sub                      (BLFL)
0990-099F -- Hand types look-up table
09A0-09AF -- Messages look-up table
09B0-09FF -- Data/Names/Bit patterns/ASCII strings
0A00-0AEB -- MLS--hand evaluation
0AEC-0AFF -- Look-up coversion table

0B00-0B15 -- MLS--exchange                                (XCHNG)
0B16-0B37 -- MLS--sort hand
0B38-0B7A -- MLS--display handsdecoding
0B7B-0B8F -- MLS--ante
0B90-0BCB -- Storage for hands & evaluations
0BCC-0BFF -- Deck of cards
```

0C14-0C1F -- MLS--set hands = FF's
0C40-0C73 -- MLS--Draw cards
0C8C-0CB2 -- Bet module control sub (entry @ 0C90) (MOD)
0CB8-0CBE -- Bet greater than cash sub (BET $)
0D6C-0DB9 -- ASCII message strings
0DBA-0DFC -- Figure bet (part of module bet sub) (FIGR)

PROGRAM LISTING

```
0300  BEGIN :297E ;BLFL  -- Do sub -- set up strategies
  02          AF38 ;00's  -- Set "I" to known zero bytes in display
  04          FB65 ;GET   -- V0-VB=00's (V9 VA VB must be set to 00's)
  06          A9E0 ;CASH  -- Set "I" to player's cash variables
  08          6032 ;V0=32 -- V0=32= $50 in hex (not to exceed 3F)
  0A          28B6  STR4  -- Do sub to store 4 bytes
  0C  LOOP1 :0230  ERASE -- Clear display pages (needed on loops back)
  0E          254A  FIELD -- Do sub -- set up display

0310  LOOP2 :6D33 ;VD=33  -- VY coordinate player's cards
  12          2676  SH0-5 -- Do sub -- display 5 blank cards
  14          AF38 ;00's  -- Set "I" to known zero bytes in display
  16          F865 ;GET   -- V0-V8 = 00's (Initializes bet totals V5-V8)
  18          253C  ANTE  -- Do sub -- collect $1 from all active players
  1A          6004 ;V0=04 -- Message #4 index (Dealer)
  1C          283E  MCENT -- Do sub -- display message
  1E          2434  DEALR -- Do sub -- advance arrow to new dealer

0320          2600  SHUFF -- Do sub -- shuffle deck of cards
  22          6000 ;V0=00 -- Message #0 index (Shuffling)
  24          283E  MCENT -- Do sub -- display message
  26          2600  SHUFF -- Do sub -- shuffle deck again
  28          6001 ;V0=01 -- Message #1 (Your Hand)
  2A          283E  MCENT -- Do sub -- display message
  2C          262A  HANDS -- Do sub -- deal hands
  2E          2648  EVAL  -- Do sub -- evaluate hands

0330          6D34 ;VD=34 -- VY coordinate user's cards
  32          26A2  PLAYR -- Do sub -- show user's cards
  34          23F4  ZERO  -- Do sub -- reset bluffing flags
  36          2452  BETTR -- Do sub -- betting, round 1
  38          604F ;V0=4F -- V0 passes value to timer sub
  3A          2620  TIMER -- Do sub -- wait before continuing
  3C          2968  COUNT -- Do sub -- count number active players
  3E          4101 ;SK≠01 -- If not only one left, skip next

0340          1386  PAY   -- Go pay off single player (others folded)
  42          3F00 ;SK=00 -- If VF=00, then bet was at least opened
  44          130C  LOOP1 -- Else go new deal -- nobody opened
  46          2442  DLR2  -- Do sub -- reset arrow to dealer
  48          284C  DRAW  -- Do sub -- draw cards (all players)
  4A          2648  EVAL  -- Do sub -- evaluate hands
  4C          2452  BETTR -- Do sub -- betting, round 2
  4E          606F ;V0=6F -- V0 passes value to timer sub
```

```
0350          2620 ;TIMER -- Do sub -- wait before continuing
  52          2968  COUNT -- Do sub -- count number active players
  54          4101 ;SK≠00 -- If not only one, skip next
  56          1386  PAY   -- Go pay off single player
  58          2648  EVAL  -- Do sub -- evaluate hands (so folders =???)
  5A          0230 ;ERASE -- Clear display for showdown
  5C          2692  DECOD -- Do sub -- decode and display all hands
  5E          26BA  RANK  -- Do sub -- display ranks of hands

0360          A9E4 ;NAMES -- Set "I" to ASCII for player names
  62          6D00 ;VD=00 -- VY coordinate for names
  64 LOOP3   :2846  PRINT -- Print one name
  66          7D10 ;VD+10 -- Add 10 to VY for next name down
  68          3D40 ;SK=40 -- But skip next when past last name
  6A          1364  LOOP3 -- Go print next name
  6C          26D8  WINER -- Do sub -- determine winner (Key 0 returns)
  6E LOOP4   :0230  ERASE -- Clear screen (showdown done)

0370          254A  FIELD -- Do sub -- set up display
  72          24F4  PAYOF -- Do sub -- pay off the winner
  74          606F ;V0=6F -- V0 passes value to timer sub
  76          2620 ;TIMER -- Do sub -- wait before continuing
  78          3E01 ;SK=01 -- If VE=01, skip to end
  7A          1310  LOOP2 -- Else loop back for next deal
  7C END     :252A  DOLAR -- Do sub -- erase/display (winner's) cash
  7E          6000 ;V0=00 -- Set V0=00 to allow restart feature

0380          E09E ;SK=KY -- If Key 0 is pressed, skip next
  82          137C  END   -- Continue to loop till Key 0 pressed
  84          1300  BEGIN -- Then restart new game
  86 PAY     :8E20 ;VE=V2 -- Let VE = single player in V2
  88          136E  LOOP4 -- Go pay off the single player
```

PLAYER BETTING MODULE

```
038A BET     :2524  CSHIN -- Do sub -- set "I" to player's cash
  8C          F065 ;GET   -- Player's cash in V0
  8E          8300 ;V3=V0 -- Save cash in V3

0390          4300 ;SK≠00 -- If not = $0, skip to continue
  92          15B4  FOLD  -- Else go fold player with no money
  94          8095 ;V0-V9 -- Subtract minimum from cash
  96          3F01 ;SK= + -- If positive, continue (cash ≥ minimum)
  98          15B4  FOLD  -- Else fold player who cannot bet minimum
  9A          8C40 ;VC=V4 -- Save player number (V4) in VC
  9C          2968  COUNT -- Do sub -- count number active players
  9E          84C0 ;V4=VC -- Reset V4 from saved value in VC
```

```
03A0          6280 ;V2=80 -- Set V2 to a constant of 80 (weight)
  A2          4102 ;SK≠02 -- If not only two players active, skip
  A4          7202 ;V2+2  -- Weight + 2 (Discourage fold)
  A6          4103 ;SK≠03 -- If not only three players, skip
  A8          7202 ;V2+2  -- Weight + 2 (Discourage fold)
  AA          AB95 ;EVAL1 -- Set "I" to betting player's hand eval
  AC          4402 ;SK≠02 --
  AE          ABA4 ;EVAL2 --          "           "

03B0          4403 ;SK≠03 --
  B2          ABB3 ;EVAL3 --          "           "
  B4          F165 ;GET   -- V0=evaluation/V1= possible pair
  B6          4000 ;SK≠00 -- If V0≠00, skip (not a zilch hand)
  B8          72FF ;V2-1  -- Weight - 1 for z'lches (V0=00)
  BA          800E ;SHL   -- Shift V0 left to multiply x 2
  BC          800E ;SHL   --   "          "          "
  BE          3004 ;SK=04 -- If eval = 04 (1 pair) skip into next

03C0          13CA  BET 1 -- Else continue past next part
  C2          632A ;V3=2A -- Let V3 = value for pair of 10's
  C4          8315 ;V3-V1 -- Subtract 2A - possible pair in V1
  C6          3F01 ;SK= + -- If possible, skip (V1≤ pair 10's)
  C8          7201 ;V2+1  -- Weight + 1 for Jacks or better
  CA          8204 ;V2-V0 -- Weight + evaluation
  CC          8295 ;V2+V9 -- Weight minimum
```

BLUFFER

```
03CE  BLUFF :A3FC ;BLFLG -- Set "I" to bluff flags

03D0          F41E ;I+V4  -- Index to betting player's flag
  D2          F065 ;GET   -- V0 = bluff flag @ M(I)
  D4          3000 ;SK=00 -- If = 00, skip into next section
  D6          13E8  BLUF2 -- Go bluff -- bluff in progress
  D8          AC0C  MASKS -- Set "I" to CXKK masks (strategies)
  DA          F41E ;I+V4  -- Index to betting player's strategy
  DC          F065 ;GET   -- V0 = bluffing mask @ M(I)
  DE          A3E3 ;BLUF1 -- Set "I" to KK in CXKK below

03E0          F055 ;PUT   -- Store mask in instruction below
  E2  BLUF1 :C0KK ;RND    -- V0=RND number, range variable
  E4          3000 ;SK=00 -- If 00 generated, skip into bluff
  E6          1DBA  FIGR  -- Go figure bet -- no bluff
  E8  BLUF2 :7206 ;V2+06  -- Weight + 06 to overrepresent   hand
  EA          6001 ;V0=01 -- Set bluff flag for this player
  EC          A3FC  BLFLG -- Set "I" to bluff flags
  EE          F41E ;I+V4  -- Index to betting player's flag
```

```
03F0          F055 ;PUT    -- Store 01 flag
  F2          1DBA  FIGR   -- Go figure bet -- bluff
```

RESET BLUFF FLAGS = 00 SUB

```
03F4  ZERO   ;A3FC ;BLFLG -- Set "I" to bluff flags
  F6          6000 ;V0=00 -- V0 passes value to store sub
  F8          28B6  STR4  -- Do sub to store contents V0
  FA          00EE ;RET    -- Return
```

BLUFF FLAGS

```
03FC  BLFLG  :0000 ;FLAGS -- Bluff flag #1 @ 03FD
  FE          0000 ;FLAG  -- Bluff flag #2 @ 03FE/#3 @ 03FF
```

ARROW SUB

```
0400  ARROW  :6C1C ;VC=1C -- VC=X coordinate arrow display
  02          6D0E ;VD=0E -- VD=Y      "       "       "
  04          A420  AINDX -- Set "I" to storage last known position
  06          F065 ;GET   -- Let V0= last known position
  08          2422  ARR01 -- Do sub -- display arrow to erase old
  0A          3F01 ;SK=01 -- If display "hit", skip next (arrow off)
  0C          2422  ARR01 -- Else redisplay to initialize first arrow
  0E          7001 ;V0+01 -- Add 01 to the arrow position index

0410          4005 ;SK≠05 -- And skip next as long as index < 5
  12          6001 ;V0=01 -- Reset index = 01 when it goes to 5
  14          2422  ARR01 -- Do sub -- display new arrow one over
  16          6101 ;V1=01 -- V1 = value for tone
  18          F118 ;TONE  -- Sound tone
  1A          A420  AINDX -- Set "I" to storage last known position
  1C          F055 ;PUT   -- Store V0 @ M(I) to save current position
  1E          00EE ;RET   -- Return -- V0=player # pointed to

0420  AINDX  :0100 ;INDEX -- Storage for last known position (Index)
                           -- Set initially = 01, but for no reason
```

ARRO1 SUB

```
0422  ARRO1  :A9B4 ; 1      -- Set "I" to bit pattern arrow #1
  24          4002 ;SK≠02   -- Skip next if V0 index ≠ 02
  26          A9BC ; 2      -- Set "I" to bit pattern arrow #2
  28          4003 ;SK≠03   -- Skip next if V0 index ≠ 03
  2A          A9C4 ; 3      -- Set "I" to bit pattern arrow #3
  2C          4004 ;SK≠04   -- Skip next if V0 index ≠ 04
  2E          A9CC ; 4      -- Set "I" to bit pattern arrow #4

0430          DCD8 ;SHOW    -- Display arrow
  32          00EE ;RET     -- Return
```

DEALER SUB

```
0434  DEALR  :A450 ;DINDX   -- Set "I" to last known dealer position
  36          F065 ;GET     -- V0 = last dealer #
  38          7001 ;V0+01   -- Add 01 to index for new dealer
  3A          4005 ;SK≠05   -- Skip next as long as index ≠ 05
  3C          6001 ;V0=01   -- Reset index to = 01 when = 05
  3E          A450 ;DINDX   -- Set "I" to storage for the index

0440          F055 ;PUT     -- Store V0 @ M(I) (For use by next part)
  42  DLR2   :A450 ;DINDX   -- Enter here to point to current dealer
  44          F065 ;GET     -- "I" = dealer index -- V0 = index value
  46          8200 ;V2=V0   -- Save the index in V2
  48  DLR3   :2400  ARROW   -- Do sub -- advance arrow (V0=position)
  4A          5200 ;=SKIP   -- When arrow position=dealer, skip next
  4C          1448  DLR3    -- Else loop back to advance arrow again
  4E          00EE ;RET     -- Return- arrow on dealer (Index in V0)

0450  DINDX  :0300 ;INDEX   -- Storage for dealer index
                            -- Initially=03 so user is first dealer
```

BETTOR SUB

```
0452  BETTR  :35FF ;SK=FF   -- V5 V6 V7 V8 = betting totals or fold flags
  54          6500 ;V5=00   -- These instructions reset V5 V6 V7 V8
  56          36FF ;SK=FF   -- To equal 00 unless they are
  58          6600 ;V6=00   -- Equal to FF (i.e. player has folded)
  5A          37FF ;SK=FF   --   "          "          "
  5C          6700 ;V7=00   --   "          "          "
  5E          38FF ;SK=FF   --   "          "          "
```

```
0460          6800 ;V8=00 -- "          "          "
  62          1474 ;BETT2 -- Jump into main part of sub
  64  BETT1 ;3900 ;SK=00 -- If minimum to bet still = $0, skip into next
  66          1474  BETT2 -- Else jump to continue betting

      ;TEST FOR PASS

  68          A450  DINDX -- Set "I" to dealer index storage byte
  6A          F065 ;GET   -- V0= M(I) = the current dealer number
  6C          5040 ;=SKIP -- If V0=V4 (and minimum=$0) all pass; skip
  6E          1474  BETT2 -- Else jump to continue betting

0470          6F01 ;VF=01 -- Set VF=01 to flag no open bets
  72          00EE ;RET   -- Then return (all players passed) (exit sub)
  74  BETT2 ;604F ;V0=4F -- V0 passes value to timer sub
  76          2620 ;TIMER -- Do sub -- wait before continuing
  78          2400  ARROW -- Do sub -- advance arrow next player
  7A          8400 ;V4=V0 -- Save player number in V4 (V0=AINDX)
  7C          2518  AMT   -- Do sub -- V0=total bet so far for player
  7E          40FF ;SK≠FF -- Skip if total ≠ FF (Player folded if = FF)

0480          1464  BETT1 -- Jump back to skip the folded player
  82          4900 ;SK≠00 -- If minimum ≠ $0, skip (bet was opened)
  84          14A8  BETT3 -- Else jump forward to skip next part

      ;FIGURE MINIMUM BET

  86          2820 ;BET$  -- Do sub -- erase betting amount displayed
  88          2518  AMT   -- Do sub -- V0=total bet by player
  8A          8100 ;V1=V0 -- Save the amount in V1
  8C          A9B2 ;TOTAL -- Set "I" to storage of last betting total
  8E          F065 ;GET   -- V0=last player to bet total bet

0490          8015 ;V0-V1 -- Subtract the totals bet
  92          8900 ;V9=V0 -- Let V9=the result -- the minimum bet
  94          2820 ;BET$  -- Do sub -- display the minimum bet

      ;BET CALLED?

  96          3900 ;SK=00 -- If minimum goes to $0, skip next
  98          14A8  BETT3 -- Jump to continue if V9≠00
  9A          6003 ;V0=03 -- Message #3 index (bet called)
  9C          283E ;MCENT -- Do sub-display message
  9E          282A ;RSE$  -- Do sub -- erase raise number
```

```
04A0          6A00 ;VA=00 -- Reset raise number to zero
  A2          282A ;RSE$  -- Do sub -- redisplay raise number (=000)
  A4          6F00 ;VF=00 -- Set VF flag=00 to signal active betting
  A6          00EE ;RET   -- Then return (exit sub)
  A8  BETT3 :6002 ;V0=02 -- Message #2 index (Please bet)
  AA          283E ;MCENT -- Do sub -- display message
  AC          2C90 ;MOD   -- Do sub -- call betting modules
                                    -- bet returned @ 09B3
  AE          A9B3 ; $    -- Set "I" to bet @ 09B3

04B0          F065 ;GET   -- Let V0 = amount being bet
  B2          40FF ;SK≠FF -- If ≠FF (folding) skip into next part
  B4          1464  BETT1 -- Else jump back for next round -- player folds
  B6          4900 ;SK≠00 -- If minimum≠ 00, skip into next part
  B8          14C8  BETT4 -- Else jump forward to skip next part
  BA          9090 ;≠SKIP -- If amount bet ≠ minimum, skip into next part
  BC          14C8 ;BETT4 -- Else jump forward (no raise)

      ;PLAYER RAISES

  BE          282A ;RSE$  -- Do sub -- erase the raise number

04C0          7A01 ;VA+01 -- Add 1 to raise number
  C2          282A ;RSE$  -- Do sub -- redisplay raise number (+01)
  C4          6005 ;V0=05 -- Message #5 index (bet raised)
  C6          283E ;MCENT -- Do sub -- display message
  C8  BETT4 :2820 ;BET$   -- Do sub -- erase minimum bet
  CA          A9B3 ; $    -- Set "I" to amount being bet
  CC          F065 ;GET   -- V0=player's bet
  CE          8900 ;V9=V0 -- Let V9 = the bet

04D0          2820 ;BET$  -- Do sub -- display player's bet

      ;COLLECT BET

04D2          252A  DOLAR -- Do sub -- erase player's cash
  D4          2524  CSHIN -- Do sub -- set "I" to player's cash storage
  D6          F065 ;GET   -- Let V0 = player's cash
  D8          8095 ;V0-V9 -- Subtract bet from cash
  DA          2524  CSHIN -- Do sub -- set "I" to player's cash storage
  DC          F055 ;PUT   -- Store V0 @ M(I) -- adjusted cash
  DE          252A  DOLAR -- Do sub -- display player's cash

04E0          2518  AMT   -- Do sub -- V0=total bet so far
  E2          8094 ;V0+V9 -- Add amount just bet (in V9)
  E4          251E  AMTIN -- Do sub -- set "I" to player's total variable
  E6          F055 ;PUT   -- Store new total as V5 V6 V7 or V8
```

```
04E8          A9B2 ;TOTAL -- Set "I" to storage byte
  EA          F055 ;PUT   -- Store total for figuring minimum
  EC          2834 ;POT$  -- Do sub -- erase pot
  EE          8B94 ;VB+V9 -- Add amount bet to VB (Pot variable)

04F0          2834 ;POT$  -- Do sub -- display new pot
  F2          1464  BETT1 -- Jump back to continue sub
```

PAYOFF SUB (VE=WINNER)

```
04F4  PAYOF ;2400  ARROW -- Do sub -- advance arrow
  F6          50E0 ;=SKIP -- Skip if V0 index = VE winner
  F8          14F4  PAYOF -- Else loop back till arrow finds winner
  FA          84E0 ;V4=VE -- Let V4=winner in VE (for next sub)
  FC          252A  DOLAR -- Do sub -- erase winner's cash
  FE          2524  CSHIN -- Do sub -- set "I" to player's cash storage

0500          F065 ;GET   -- Let V0 = player's cash
  02          80B4 ;V0+VB -- Add pot to cash
  04          2524  CSHIN -- Do sub -- set "I" to player's cash storage
  06          F055 ;PUT   -- Store new cash value (+ Pot)
  08          6E00 ;VE=00 -- VE = end game flag (00=continue)
  0A          40C8 ;SK≠C8 -- Skip if V0≠C8=$200 (may be changed)
  0C          6E01 ;VE=01 -- VE=end game flag (01=stop game)
  0E          252A  DOLAR -- Do sub -- display player's cash

0510          2834  POT$  -- Do sub -- erase pot
  12          6B00 ;VB=00 -- Set pot variable VB to 00
  14          2834  POT$  -- Do sub -- display pot (=000)
  16          00EE ;RET   -- Return
```

AMOUNT SUB

```
0518  AMT    ;251E  AMTIN -- Do sub -- set "I" to player's total
  1A          F065 ;GET   -- Let V0=M(I)= total bet/fold flag
  1C          00EE ;RET   -- Return

  1E  AMTIN ;A2F4 ;AMT-1 -- Set "I" to Chip-8 V5-1 (V4)

0520          F41E ;I+V4  -- Add player # in V4 to I
  22          00EE ;RET   -- Return- "I" set to V5 V6 V7 or V8 bytes
```

CASH INDEX SUB

```
0524  CSHIN :A9DF ;CSH-1 -- Set "I" to player's cash byte - 01
  26         F41E ;I+V4  -- Add player # in V4 to I
  28         00EE ;RET   -- Return- "I" set to player's cash
```

DOLLAR SUB

```
052A  DOLAR :4401 ;SK≠01 -- Skip next if V4 (Player #) ≠ 01
  2C         27DE ;MONY1 -- Do sub -- display/erase player #1 cash
  2E         4402 ;SK≠02 -- Skip next if V4 (Player #) ≠ 02

0530         27E8 ;MONY2 -- Do sub -- display/erase player #2 cash
  32         4403 ;SK≠03 -- Skip next if V4 (Player #) ≠ 03
  34         27F2 ;MONY3 -- Do sub -- display/erase player #3 cash
  36         4404 ;SK≠04 -- Skip next if V4 (Player #) ≠ 04
  38         27FC ;MONY4 -- Do sub -- display/erase player #4 cash
  3A         00EE ;RET   -- Return
```

ANTE SUB

```
053C  ANTE  :2834 ;POT$  -- Do sub -- erase pot
  3E         255A  CASH  -- Do sub -- erase all players cash

0540         A9E0 ; 1    -- Set "I" to players' cash storage
  42         0B7B ;MLS   -- Do MLS -- collect $1 ante to play
  44         255A  CASH  -- Do sub -- display all players cash
  46         2834 ;POT$  -- Do sub -- display new pot (+ antes)
  48         00EE ;RET   -- Return
```

FIELD SUB (DISPLAY SET-UP)/CASH SUB

```
054A  FIELD :27B6 ;NAME1 -- Do sub -- display player #1 name
  4C         27C0 ;NAME2 -- Do sub --     "        "   #2  "
  4E         27CA ;NAME3 -- Do sub --     "        "   #3  "

0550         27D4 ;NAME4 -- Do sub --     "        "   #4  "
  52         280C ;BRP   -- Do sub -- display words "Bet; Rse; Pot"
  54         2820 ;BET$  -- Do sub -- display bet amount
```

```
0556          282A ;RSE$    -- Do sub -- display raise number
  58          2834 ;POT$    -- Do sub -- display pot amount
  5A  CASH :27DE ;MONY1     -- Do sub -- display player #1 cash
  5C          27E8 ;MONY2   -- Do sub -- display player #2 cash
  5E          27F2 ;MONY3   -- Do sub -- display player #3 cash

0560          27FC ;MONY4   -- Do sub -- display player #4 cash
  62          00EE ;RET     -- Return
```

BLINK SUB (USER DRAWS CARDS)

```
0564  RSTR  :6007 ;V0=07 -- Message #7 index (Try Again)
  66          283E ;MCENT -- Do sub -- Display message
  68  BLINK :6200 ;V2=00 -- Begin sub here--V2=number cards drawn
  6A  BLIN1 :F30A ;V3=KY -- Let V3=value hex key pressed (Waits)
  6C          430E ;SK≠0E -- Skip next if Key E (Enter) not pressed
  6E          159E  BLIN2 -- Go end on Key E

0570          73FF ;V3-01 -- Subtract 01 (= to + FF) from key pressed
  72          6004 ;V0=04 -- Let V0=04 to test range
  74          8035 ;V0-V3 -- Subtract 4- key pressed (If +, key≤04)
  76          3F01 ;SK=+  -- Skip if positive (Key 1, 2, 3, 4, or 5
                                                          selected
  78          1564  RSTR  -- Else jump up to restart/error message
  7A          4203 ;SK≠03 -- Skip if number drawn ≠ 03 yet
  7C          1564  RSTR  -- Else jump up to restart/error message
  7E          8C30 ;VC=V3 -- VC=VX coordinate marker display

0580          8CCE ;SHL   -- Shifting multiplies x 2
  82          8CCE ;SHL   --    "          "       "
  84          8CCE ;SHL   --    "          "       " (x 8 total)
  86          6D30 ;VD=30 -- VD=VY coordinate marker display
  88          AC3A ;MARKR -- Set "I" to bit pattern of marker
  8A          DCD2 ;SHOW  -- Display marker above card to discard
  8C          3F00 ;SK=00 -- Skip next if marker did not hit another
  8E          1564  RSTR  -- Else jump up to restart -- same card
                                                        discarded
0590          ABBD ;HAND4 -- Set "I" to user's cards in memory
  92          F31E ;I+V3  -- Add key pressed (-01) to "I"
  94          F065 ;GET   -- V0= card to discard
  96          8100 ;V1=V0 -- V1 passes card to next sub
  98          25A4  DRAW  -- Do sub -- insert card in user's eval
  9A          7201 ;V2+01 -- Add one to V2 to count the discard
  9C          156A  BLIN1 -- Loop back for next instruction
  9E  BLIN2 :61FF ;V1=FF -- End sub -- V1 passes FF stop byte to sub
```

```
05A0          25A4  DRAW   -- Do sub -- Insert FF byte after discards
  A2          00EE ;RET    -- Return
```

DRAW SUB

```
05A4  DRAW   :ABC2 ;EVAL4  -- Set "I" to user's evaluation area
  A6  DRAW1  :F065 ;GET    -- Let V0 = a byte from eval ("I"+1)
  A8          30FF ;SK=FF  -- Skip if FF byte found
  AA          15A6  DRAW1  -- Else loop back till "I" properly set
  AC          F21E ;I+V2   -- Add # cards drawn to "I"
  AE          8010 ;V0=V1  -- Retrieve passed value in V1

05B0          F055 ;PUT    -- Store V0 @ M(I) to discard
  B2          00EE ;RET    -- Return
```

FOLD SUB (FOLDING MODULES JUMP TO HERE)

```
05B4  FOLD   :3401 ;SK=01  -- Skip into next for player #1
  B6          15BE  FOLD1  -- Else go to next section
  B8          65FF ;V5=FF  -- Set player total = FF
  BA          27B6 ;NAME1  -- Do sub -- erase player name #1
  BC          AB90 ;HAND1  -- Set "I" to player #1 cards
  BE  FOLD1  :3402         --  Above

05C0          15C8  FOLD2  --    Comments
  C2          66FF ;V6=FF  --      Apply
  C4          27C0 ;NAME2  --        to following
  C6          AB9F ;HAND2  --  But
  C8  FOLD2  :3403 ;SK=03  --    For
  CA          15D2  FOLD3  --      Players
  CC          67FF ;V7=FF  --        #2, 3, and 4
  CE          27CA ;NAME3  --

05D0          ABAE ;HAND3  --        "           "
  D2  FOLD3  :3404 ;SK=04  --        "           "
  D4          15DC  FOLD4  --        "           "
  D6          68FF ;V8=FF  --        "           "
  D8          27D4 ;NAME4  --        "           "
  DA          ABBD ;HAND4  --        "           "
  DC  FOLD4  :0C14 ;MLS    -- Do MLS ("I" preset) set player's hand to FF's
  DE          ABC0 ;FOLDS  -- Set "I" to ASCII string for "FOLDS"
```

```
05E0          2846 ;PRINT -- Do sub -- print "FOLDS" in place of name
  E2          61FF ;V1=FF -- V1 passes FF fold byte to next section
  E4  STRBT :8010 ;V0=V1 -- V0 = value in V1 (all bet mods jump here)
  E6          A9B3 ; $    -- Set "I" to betting storage byte
  E8          F055 ;PUT   -- Store amount bet (or FF folds byte)@ M(I)
  EA          00EE ;RET   -- Return (all betting subs return from here)

05EC-05FF -- Unused -- Set to 00's
```

SHUFFLE SUB

```
0600  SHUFF :6EFF ;VE=FF -- Loop count of FF in VE
  02  SHUF1 :2614  GENER -- Do sub -- generate random # 00-33 hex
  04          8100 ;V1=V0 -- Save number generated in V1
  06          2614  GENER -- Do sub -- generate random # 00-33 hex
  08          ABCC ;CARDS -- Set "I" to deck of cards @ 0B00
  0A          0B00 ;MLS   -- Do MLS -- exchange two cards in the deck
  0C          7EFF ;VE-01 -- Loop count -01 by adding FF
  0E          3E00 ;SK=00 -- Skip when VE goes to 00

0610          1602  SHUF1 -- Else continue to shuffle
  12          00EE ;RET   -- Return (in about 4 seconds)
```

GENERATE RANDOM # 00-33

```
0614  GENER :C03F ;RND    -- V0=RND # from 00-3F hex
  16          6F33 ;VF=33 -- VF = limit of 33 hex
  18          8F05 ;VF-V0 -- Subtract 33-RND # (if neg., RND # >33)
  1A          3F01 ;SK=+  -- Skip if result positive or zero
  1C          1614  GENER -- Else loop back for another number
  1E          00EE ;RET   -- Return (number in V0)
```

TIMER SUB

```
0620  TIMER :F015 ;TI=V0 -- Set timer to value in V0 (passed by caller)
  22  TIME  :F007 ;V0=TI -- Let V0 = current timer value
  24          3000 ;SK=00 -- When timer = 00 (in 1/60 x V0 seconds)
  26          1622  TIME  -- Loop back till timer goes to 00
  28          00EE ;RET   -- Then return
```

HANDS SUB (DEALS HANDS)

```
062A  HANDS :6ECC ;VE=CC -- VE is initialized to first card address
  2C         2660  CARDS -- Do sub -- get next 5 cards in V0-V4
  2E         AB90 ;HAND1 -- Set "I" to player's hand #1

0630         F455 ;PUT   -- Store 5 cards in player's hand
  32         2660  CARDS -- Repeat above
  34         AB9F ;HAND2 --   For players
  36         F455 ;PUT   --     #2, 3, and 4
  38         2660  CARDS --       "                "
  3A         ABAE ;HAND3 --       "                "
  3C         F455 ;PUT   --       "                "
  3E         2660  CARDS --       "                "

0640         ABBD ;HAND4 --       "                "
  42         F455 ;PUT   --       "                "
  44         1646 ;NOP   -- No operation, go to next instruction
  46         00EE ;RET   -- Return
```

EVALUATION SUB

```
0648  EVAL  :AB90 ;HAND1 -- Set "I" to player's hand #1
  4A         265A  EVAL1 -- Do sub -- calls MLS evaluation routines
  4C         AB9F ;HAND2 -- Repeat for
  4E         265A  EVAL1 --  Player hands

0650         ABAE ;HAND3 --    #2, 3, and 4
  52         265A  EVAL1 --      "                "
  54         ABBD ;HAND4 --      "                "
  56         265A  EVAL1 --      "                "
  58         00EE ;RET   -- Return

  5A  EVAL1 :0B16 ;MLS   -- Do MLS -- sort one hand (in order)
  5C         0A00 ;MLS   -- Do MLS -- evaluate one hand
  5E         00EE ;RET   -- Return
```

CARDS SUB (DEALS 5 CARDS INTO V0-V4)

```
0660  CARDS :AB00 ;DECK  -- Set "I" to base page address of deck
  62         FE1E ;I+VE  -- Add address index in VE to I (finds deck top)
  64         F465 ;GET5  -- Let V0-V4 = memory bytes @ I
  66         7E05 ;VE+05 -- Add 5 to VE index for next call
  68         00EE ;RET   -- Return (5 cards in V0-V4)
```

BLANK SUB

```
066A  BLANK :6D00 ;VD=00 -- VD is VY coordinate for showdown display
  6C  BLAN1 :2676  SH0-5 -- Do sub -- display 5 blank cards
  6E         7D10 ;VD+10 -- Add 10 hex to VD for next row cards

0670         3D40 ;SK=40 -- Skip when VD=40 (5 rows shown)
  72         166C  BLAN1 -- Go loop till VD = 40
  74         00EE ;RET   -- Return (blank showdown cards shown)
```

SHOW FIVE SUB

```
0676  SH0-5 :6C00 ;VC=00 -- VC is VX coordinate for blank cards display
  78  SH0   :A684  BLOCK -- Set "I" to bit pattern for blank card
  7A         DCDD ;SHOW  -- Display a blank card @ VC VD
  7C         7C08 ;VC+08 -- Add 08 to VX coordinate for next card
  7E         3C28 ;SK=28 -- But skip when Vc=28 (5 cards shown)

0680         1678  SH0   -- Go loop till VC=28
  82         00EE ;RET   -- Return (one row 5 cards displayed)

  84  BLOCK :FEFE ;BITS  -- Bit pattern for solid blank card
  86         FEFE        --   "                    "
  88         FEFE        --   "                    "
  8A         FEFE ; "    --   "                    "
  8C         FEFE        --   "                    "
  8E         FEFE        --   "                    "
0690         FE00 ; "    --   "                    "
```

DECODE SUB

```
0692  DECOD :266A  BLANK -- Do sub -- display 5 rows blank cards
  94         6D01 ;VD=01 -- VD is VY coordinate for display
  96         AB90 ;HAND1 -- Set "I" to player's hand #1
  98         26A8  SHOCD -- Do sub -- decode and display card values
  9A         AB9F ;HAND2 -- Repeat for
  9C         26A8  SHOCD --     hands
  9E         ABAE ;HAND3 --        #2, 3 and 4

06A0         26A8  SHOCD --    "                 "                "
  A2  PLAYR :ABBD ;HAND4 --    " (enter for displaying user hand) "
  A4         26A8  SHOCD --    "                 "                "
  A6         00EE ;RET   -- Return
```

SHOW CARD SUB

```
06A8  SHOCD :0B38 ;MLS    -- Do MLS -- decodes hands into ASCII equivalents
  AA          6C00 ;VC=00  -- VC is VX coordinate for display
  AC          AC24 ;STORE  -- Set "I" to 21 byte work area @ 0C24
  AE          2846  PRINT  -- Do sub -- print hand types (2's, 4's etc.)

06B0          7C02 ;VC+02  -- Add 2 to VX coordinate (which was not changed)
  B2          7D06 ;VD+06  -- Add 6 to VY     "                  "
  B4          2846  PRINT  -- Do sub -- print suits (clubs, hearts, etc.)
  B6          7D0A ;VD+0A  -- Add 0A to VY coordinate for next call
  B8          00EE ;RET    -- Return
```

RANK SUB

```
06BA  RANK  :6C28 ;VC=28  -- VC is VX for rank display (1 pair, fulhse,etc.)
  BC          6D07 ;VD=07  -- VD is VY    "                    "
  BE          AB95 ;EVAL1  -- Set "I" to evaluation #1 (Player #1)

06C0          26D0  SET I  -- Do sub -- Set "I" to ASCII for  that hand
  C2          ABA4 ;EVAL2  -- Repeat for                type and print
  C4          26D0  SET I  --     hands
  C6          ABB3 ;EVAL3  --         #2, 3 and 4
  C8          26D0  SET I  --    "                 "
  CA          ABC2 ;EVAL4  --    "                 "
  CC          26D0  SET I  --    "                 "
  CE          00EE ;RET    -- Return
```

SET "I" SUB

```
06D0  SET I :0959 ;MLS    -- Do MLS -- "I" set to ASCII string (fulhse, etc.)
  D2          2846  PRINT  -- Do sub -- print the hand type @ VC VD
  D4          7D10 ;VD+10  -- Add 10 to VY coordinate for next call
  D6          00EE ;RET    -- Return
```

WINNER SUB

```
06D8  WINER :AB95 ;EVAL1  -- Set "I" to evaluation #1
  DA          0B6A ;MLS    -- Do MLS -- set V1 V2 V3 V4= evaluations
```

```
06DC          0900 ;MLS    -- Do MLS -- eliminate all but highest eval(s)
  DE          6E01 ;VE=01  -- VE counts the number passes through loop

06E0  WIN    :4100 ;SK≠00  -- Skip if V1≠00 (not eliminated)
  E2          16EC  WIN1   -- Go to next part
  E4          AB96 ;PR.1   -- Set "I"
  E6          2926  PICK   -- Do sub -- pick up player's pairs
  E8          3FFF ;SK=FF  -- Skip next if VF (returned by sub)=FF (no pairs)
  EA          81F0 ;V1=VF  -- Let V1=VF= pair for comparing
  EC  WIN1   :4200 ;SK≠00  -- Repeat above
  EE          16F8  WIN2   --     6 instructions

06F0          ABA5 ;PR.2   --          but for players
  F2          2926  PICK   --               #2, 3 and 4
  F4          3FFF ;SK=FF  --  "                    "
  F6          82F0 ;V2=VF  --  "                    "
  F8  WIN2   :4300 ;SK≠00  --  "                    "
  FA          1704  WIN3   --  "                    "
  FC          ABB4 ;PR.3   --  "                    "
  FE          2926  PICK   --  "                    "

0700          3FFF ;SK=FF  --  "                    "
  02          83F0 ;V3=VF  --  "                    "
  04  WIN3   :4400 ;SK≠00  --  "                    "
  06          1710  WIN4   --  "                    "
  08          ABC3 ;PR.4   --  "                    "
  0A          2926  PICK   --  "                    "
  0C          3FFF ;SK=FF  --  "                    "
  0E          84F0 ;V4=VF  --  "                    "

0710  WIN4   :0900 ;MLS    -- Do MLS -- eliminate all but highest pair(s)
  12          7EFF ;VE-01  -- Subtract 01 from pass counter VE
  14          3EFF ;SK=FF  -- If VE goes to FF (past 00) skip to next part
  16          16E0  WIN    -- Else loop back to do possible 2nd pair
  18          6E04 ;VE=04  -- Begin high card in hand elimination
  1A  WIN9   :4100 ;SK≠00  -- Skip into next if V1≠00 (player #1 not
  1C          1728  WIN5   -- Else jump to next section      eliminated)
  1E          AB90 ;HAND1  -- Set "I" to player's hand #1

0720          FE1E ;I+VE   -- Add VE index to "I" (cards compared last
  22          F065 ;GET    -- V0 = M(I) = card in hand          to first)
  24          610F ;V1=0F  -- V1 = 0F
  26          8102 ;V1&V0  -- Logical AND card into V1 to strip suit
  28  WIN5   :4200 ;SK≠00  -- Repeat above
  2A          1736  WIN6   --      7 lines
  2C          AB9F ;HAND2  --          but for
  2E          FE1E ;I+VE   --               hands #2, 3 and 4
```

```
0730          F065 ;GET    --  "              " (see preceding page)
  32          620F ;V2=0F  --  "              "
  34          8202 ;V2&V0  --  "              "
  36  WIN6   :4300 ;SK≠00  --  "              "
  38          1744  WIN7   --  "              "
  3A          ABAE ;HAND3  --  "              "
  3C          FE1E ;I+VE   --  "              "
  3E          F065 ;GET    --  "              "

0740          630F ;V3=0F  --  "              "
  42          8302 ;V3&V0  --  "              "
  44  WIN7   :4400 ;SK≠00  --  "              "
  46          1752  WIN8   --  "              "
  48          ABBD ;HAND4  --  "              "
  4A          FE1E ;I+VE   --  "              "
  4C          F065 ;GET    --  "              "
  4E          640F ;V4=0F  --  "              "

0750          8402 ;V4&V0  --  "              "
  52  WIN8   :0900 ;MLS    -- Do MLS -- eliminate all but highest card(s)
  54          7EFF ;VE-01  -- Subtract 01 from VE index
  56          3EFF ;SK=FF  -- But skip next if index goes past 00 to FF hex
  58          171A  WIN9   -- Loop back till all cards check (end elimination)

      ;SET VE = WINNER

075A          6E01 ;VE=01  -- VE=01 = Player #1 is winner
  5C          3200 ;SK=00  -- Skip if V2=00 (not Player #2)
  5E          6E02 ;VE=02  -- VE=02 = Player #2 is winner

0760          3300 ;SK=00  -- Skip if V3=00 (not Player #3)
  62          6E03 ;VE=03  -- VE=03 = Player #3 is winner
  64          3400 ;SK=00  -- Skip if V4=00 (not Player #4)
  66          6E04 ;VE=04  -- VE=04 = Player #4 is winner

      ;DO WIN BLOCK

0768          6D06 ;VD=06  -- VD is VY coordinate win block display
  6A          4E02 ;SK≠02  -- Skip if winner ≠ 02
  6C          6D16 ;VD=16  -- Set VY
  6E          4E03 ;SK≠03  -- Skip if winner ≠ 03

0770          6D26 ;VD=26  -- Set VY
  72          4E04 ;SK≠04  -- Skip if winner ≠ 04
  74          6D36 ;VD=36  -- Set VY

  76  FLASH  :6C28 ;VC=28  -- VC is VX coordinate for win block display
```

```
0778          A790   FFS    -- Set "I" to win block bit pattern
  7A  FLSH1  :DCD7  ;SHOW   -- Display one portion (of 3) of block
  7C          7C08  ;VC+08  -- Add 08 to VX coordinate (moves right)
  7E          3C40  ;SK=40  -- But skip if VC goes to 40 hex

0780          177A   FLSH1  -- Loop back to display whole block
  82          6010  ;V0=10  -- V0 passes value to timer
  84          2620   TIMER  -- Do sub -- wait between blocks
  86          6000  ;V0=00  -- Set V0=00 for key press test
  88          E09E  ;SK=KY  -- Skip next if Key 0 is pressed
  8A          1776   FLASH  -- Else loop back to display/erase win block
  8C          178E  ;NOP    -- No operation - go to next instruction
  8E          00EE  ;RET    -- Return -- Key 0 begins new round

0790  FFS    :FFFF  ;BITS   -- Bit pattern for win block
  92          FFFF          --  "                    "
  94          FFFF          --  "                    "
  96          FF00          --  "                    "
```

NUMBERS SUB

```
0798  WORK   :0000 ;4 BYT  -- 4 byte work area for number conversion
  9A          0000
  9C  NUMBS  :A798  WORK   -- Set "I" to work area above
  9E          F033 ;3-DD   -- Convert value in V0 to 3 digit decimal number

07A0          F265 ;GET    -- Pick up digits in V0 V1 V2
  A2          6330 ;V3=30  -- Set V3=30 (base ASCII for all numbers)
  A4          8031 ;V0/V3  -- "OR" V0 with V3 (digit #1)
  A6          8131 ;V1/V3  -- "OR" V1 with V3 (digit #2)
  A8          8231 ;V2/V3  -- "OR" V2 with V3 (digit #3)
  AA          6300 ;V3=00  -- Set V3=00 null (needed by messager)
  AC          A798  WORK   -- Set "I" to work space above
  AE          F355 ;PUT    -- Store V0:V3 @ I (ASCII numbers)

07B0          A798  WORK   -- Set "I" to work space above
  B2          2846  PRINT  -- Do sub -- print 3 digit number @ VC VD
  B4          00EE ;RET    -- Return (V0 displayed or erased)
```

NAME PRINTING SUBS

```
07B6  NAME1  :6C00 ;VC=00  -- VC is VX coordinate for display
  B8          6D10 ;VD=10  -- VD is VY    "                "
```

```
07BA          A9E4 ; 1     -- Set "I" to ASCII string of name #1
  BC          2846  PRINT  -- Do sub -- print/erase name #1
  BE          00EE ;RET    -- Return

07C0  NAME2 :6C14 ;VC=14   -- VC is VX coordinate for display
  C2          6D00 ;VD=00  -- VD is VY     "              "
  C4          A9EB ; 2     -- Set "I" to ASCII string of name #2
  C6          2846  PRINT  -- Do sub -- print/erase name #2
  C8          00EE ;RET    -- Return
  CA  NAME3 :6C28 ;VC=28   -- VC is VX coordinate for display
  CC          6D10 ;VD=10  -- VD is VY     "              "
  CE          A9F2 ; 3     -- Set "I" to ASCII string of name #3

07D0          2846  PRINT  -- Do sub -- print/erase name #3
  D2          00EE ;RET    -- Return
  D4  NAME4 :6C28 ;VC=28   -- VC is VX coordinate for display
  D6          6D33 ;VD=33  -- VD is VY     "              "
  D8          A9F9 ; 4     -- Set "I" to ASCII string of name #4
  DA          2846  PRINT  -- Do sub -- print/erase name #4
  DC          00EE ;RET    -- Return
```

MONEY PRINTING SUBS

```
07DE  MONY1 :6C06 ;VC=06   -- VC is VX coordinate for cash display
  E0          6D16 ;VD=16  -- VD is VY     "                "
  E2          A9E0 ;$1$    -- Set "I" to player 1 cash
  E4          2806  DO$    -- Do sub -- get and display cash
  E6          00EE ;RET    -- Return
  E8  MONY2 :6C1A ;VC=1A   -- VC is VX coordinate for cash display
  EA          6D06 ;VD=06  -- VD is VY     "                "
  EC          A9E1 ;$2$    -- Set "I" to player 2 cash
  EE          2806  DO$    -- Do sub -- get and display cash

07F0          00EE ;RET    -- Return
  F2  MONY3 :6C2E ;VC=2E   -- VC is VX coordinate for cash display
  F4          6D16 ;VD=16  -- VD is VY     "                "
  F6          A9E2 ;$3$    -- Set "I" to player 3 cash
  F8          2806  DO$    -- Do sub -- get and display cash
  FA          00EE ;RET    -- Return
  FC  MONY4 :6C2E ;VC=2E   -- VC is VX coordinate for cash display
  FE          6D39 ;VD=39  -- VD is VY     "                "

0800          A9E3 ;$4$    -- Set "I" to player 4 cash
  02          2806  DO$    -- Do sub -- get and display cash
  04          00EE ;RET    -- Return
  06  DO$     :F065 ;GET   -- V0=cash stored @ I (set by caller)
  08          279C  NUMBS  -- Do sub -- convert and print value in V0
  0A          00EE ;RET    -- Return
```

BRP SUB (BRP IS "MNEMONIC" FOR BET RAISE POT)

```
080C  BRP    :A9D4 ;BET    -- Set "I" to ASCII string for "BET RSE POT"
  0E          6C00 ;VC=00  -- VC is VX coordinate for display "BET"

0810          6D00 ;VD=00  -- VD is VY    "              "
  12          2846  PRINT  -- Do sub -- print one ASCII string ("I" changed)
  14          6C34 ;VC=34  -- VC is VX coordinate for display "RSE"
  16          2846  PRINT  -- Do sub -- print one ASCII string ("I" changed)
  18          6C1A ;VC=1A  -- VC is VX coordinate for display "POT"
  1A          6D1B ;VD=1B  -- VD is VY    "              "
  1C          2846  PRINT  -- Do sub -- print one ASCII string
  1E          00EE ;RET    -- Return
```

BRP VALUE SUBS

```
0820  BET$   :6C00 ;VC=00  -- VC is VX coordinate for number display
  22          6D06 ;VD=06  -- VD is VY    "                    "
  24          8090 ;V0=V9  -- V0 passes value for display to sub
  26          279C  NUMBS  -- Do sub -- convert/display value in V0
  28          00EE ;RET    -- Return
  2A  RSE$   :6C34 ;VC=34  -- VC is VX coordinate for number display
  2C          6D06 ;VD=06  -- VD is VY    "                    "
  2E          80A0 ;V0=VA  -- V0 passes value for display to sub

0830          279C  NUMBS  -- Do sub -- convert/display value in V0
  32          00EE ;RET    -- Return
  34  POT$   :6C1A ;VC=1A  -- VC is VX coordinate for number display
  36          6D21 ;VD=21  -- VD is VY    "                    "
  38          80B0 ;V0=VB  -- V0 passes value for display to sub
  3A          279C  NUMBS  -- Do sub -- convert/display value in V0
  3C          00EE ;RET    -- Return
```

MESSAGE CENTER SUB/PRINT SUB

```
083E  MCENT  :093A ;MLS    -- Do MLS -- Clear message area (erasing old
                                                          message)
0840          094A ;MLS    -- Do MLS -- set "I" to ASCII message indexed
  42          6C00 ;VC=00  -- Vc is VX coordinate for display    /by V0
  44          6D2A ;VD=2A  -- VD is VY    "                    "
  46  PRINT  :0244 ;MLS    -- Do MLS -- Call messager (prints message)
  48          DCD5 ;SHOW   -- DXYN instruction for use by messager
  4A          00EE ;RET    -- Return
```

DRAW CARDS CONTROLLER SUB

```
084C  DRACD :2400 ;ARROW -- Do sub -- advance arrow to next player
  4E        8400 ;V4=V0  -- Save player number in V4 (V0 set by arrow
                                                                 sub)
0850        2518 ;AMT    -- Do sub -- V0=amount  bet/fold flag
  52        40FF ;SK≠FF  -- Skip next if flag ≠ FF (player not folded)
  54        188C  DRAC2  -- Go to end (to exit or continue)
  56        6006 ;V0=06  -- Message #6 index (CARDS?)
  58        283E  MCENT  -- Do sub -- display message
  5A        89E0 ;V9=VE  -- Save deck address index in V9
  5C        AB90 ;HAND1  -- Set "I" to player's hand #1
  5E        4402 ;SK≠02  -- Skip next if ≠ 02 (not player 2)

0860        AB9F ;HAND2  -- Set "I" to player's hand #2
  62        4403 ;SK≠03  -- Skip next if ≠ 03 (not player 3)
  64        ABAE ;HAND3  -- Set "I" to player's hand #3
  66        3404 ;SK=04  -- Skip into next section if V4=04 (user)
  68        1872  DRAC1  -- Jump past next part for players 1,2, and 3
  6A        2568 ;BLINK  -- Do sub -- user draws cards
  6C        6D34 ;VD=34  -- VD is VY for user's cards
  6E        26A2  PLAYR  -- Do sub -- erase user's cards

0870        ABBD ;HAND4  -- Set "I" to user's hand in memory
  72  DRAC1 :0C40 ;MLS    -- Do MLS -- draw cards (replaces cards
                                        discarded)
  74        2648  EVAL   -- Do sub -- evaluate hands
  76        6D34 ;VD=34  -- VD is VY coordinate for user's cards display
  78        4404 ;SK≠04  -- Skip next if V4≠04 (not user's turn)
  7A        26A2  PLAYR  -- Do sub -- display user's cards (only on
                                        your turn)
  7C        80E0 ;V0=VE  -- Let V0 = new (after draw) deck address index
  7E        8095 ;V0-V9  -- Subtract new-old indexes (V0 = # cards drawn)

0880        ABFF ;DRAWS  -- Set "I" to 4 byte store @ 0C00-01 byte
  82        F41E ;I+V4   -- Add V4 to I to index a byte for this player
  84        F055 ;PUT    -- Store # cards drawn @ I
  86        6C34 ;VC=34  -- VC is VX coordinate for number display
  88        6D2A ;VD=2A  -- VD is VY    "                    "
  8A        279C  NUMBS  -- Do sub -- display V0 @ VC VD (number drawn)
  8C  DRAC2 :6080 ;V0=80  -- V0 passes value to timer sub
  8E        2620  TIMER  -- Do sub -- wait before continuing

0890        093A ;MLS    -- Do MLS -- clear message line
  92        A450 ;DINDX  -- Set "I" to dealer index storage byte
  94        F065 ;GET    -- V0 = current dealer number
  96        5040 ;=SKIP  -- Skip if V0=V4 (when all player's done)
  98        184C  DRACD  -- Else jump back for next player
  9A        6900 ;V9=00  -- Reset V9 to 00 for use as bet variable
  9C        00EE ;RET    -- Return
```

LEGAL BET TEST SUB

```
089E  LEGAL :8010 ;V0=V1 -- Let V0 = amount user bets (passed in V1)
  A0        8095 ;V0-V9 -- Subtract V0 - minimum to bet
  A2        3F01 ;SK=+  -- Skip if result positive (V0 ≥ minimum)
  A4        00EE ;RET   -- Return with VF=00 -- signals under betting
  A6        9190 ;≠SKIP -- Skip if user bet ≠ minimum (must be greater)
  A8        00EE ;RET   -- Else return--bet=minimum VF=01 signals OK
  AA        6F00 ;VF=00 -- Set VF=00 for next parts
  AC        4A03 ;SK≠03 -- Skip if VA, raise #, ≠ 03 (must be less)
  AE        00EE ;RET   -- Return with VF=00--signals illegal raise

08B0        6F03 ;VF=03 -- Set VF=03 for next limit test
  B2        8F05 ;VF-V0 -- Subtract 03 - user bet (if neg., bet>3)
  B4        00EE ;RET   -- Return, VF=01 = good bet
                                   VF=00 = illegal raise or open
```

STORE V0 @ I SUB

```
08B6  STR4  :F055 ;PUT   -- Store V0 @ I (I+1) -- Sets 4 bytes
  B8        F055 ;PUT   --      "        "    -- to the value
  BA        F055 ;PUT   --      "        "    -- in V0, "I" set
  BC        F055 ;PUT   --      "        "    -- by caller
  BE        00EE ;RET   -- Return
```

HAND TYPES ASCII STRINGS

```
08C0        464F        -- ASCII -- "FOLDS"
  C2        4C44
  C4        5300
  C6        5A49        -- ASCII -- "ZILCH"
  C8        4C43
  CA        4800
  CC        3120        -- ASCII -- "1 PAIR"
  CE        5041

08D0        4952
  D2        0032        -- ASCII -- "2 PAIR"
  D4        2050
  D6        4149
  D8        5200
  DA        5448        -- ASCII -- "THREES"
```

```
08DC          5245
  DE          4553

08E0          0053          -- ASCII -- "STRT" (straight)
  E2          5452
  E4          5400
  E6          464C          -- ASCII -- "FLUSH"
  E8          5553
  EA          4800
  EC          4655          -- ASCII -- "FULHSE" (full house)
  EE          4C48

08F0          5345
  F2          0046          -- ASCII -- "FOURS"
  F4          4F55
  F6          5253
  F8          0053          -- ASCII -- "STRTFL" (straight flush)
  FA          5452
  FC          5446
  FE          4C00
```

MLS - HIGH SUB (V1 V2 V3 V4 = 00's OR HIGHEST VALUE)

```
0900  22  DEC   R2 ;Stack free
  01  F8  LDI
  02  F1
  03  A7  PLO   R7 ;R7.0=F1 (points to Chip-8 V1 value)
  04  87  GLO   R7          (loop here on M(R(7))>M(R(6))
  05  A6  PLO   R6 ;R6.0=R7.0
  06  06  LDN   R6 ;D=M(R(6))
  07  52  STR   R2 ;Push for comparing
  08  17  INC   R7 ;R7=R7+1  (loop here on M(R(7))≤M(R(6))
  09  87  GLO   R7
  0A  FB  XRI      ;Test if R7 past V4 value
  0B  F5
  0C  32  BZ       ;If so, exit the loop
  0D  14
  0E  07  LDN   R7 ;D=M(R(7))
  0F  F5  SD       ;Subtract M(R(x))-D ( M(R(6)) - M(R(7)) )

0910  3B  BM       ;If neg, (i.e. M(R(7))>M(R(6)) ) branch to
  11  04                    reset R(6)
  12  30  BR       ;Else branch to continue the search
  13  08
  14  F8  LDI      ;Begin set all others = 00
  15  F1           ;D=F1 (Chip-8's V1 value)
```

```
0916  A6  PLO   R6 ;R6.0=D (points to V1) (Highest byte on stack)
  17  06  LDN   R6 ;D=M(R(6))
  18  F3  XOR      ;Compare with top of stack
  19  32  BZ       ;If=, then skip next part
  1A  1E
  1B  F8  LDI      ;(Value must be < byte on stack if ≠)
  1C  00
  1D  56  STR   R6 ;M(R(6))=00 (store 00 to erase old value)
  1E  16  INC   R6
  1F  86  GLO   R6 ;D=R6.0

0920  FB  XRI
  21  F5           ;Test if R6 past V4 value
  22  3A  BNZ      ;If not, then continue
  23  17
  24  12  INC   R2 ;Reset stack pointer
  25  D4  SEP   R4 ;Return to Chip-8 control
```

PICK UP PAIRS SUB

```
0926  PICK  ;F065 ;GET   -- "I" preset by caller--V0=possible pair in
  28         8F00 ;VF=V0 -- VF=the pair in V0                      eval
  2A         40FF ;SK≠FF -- But if=FF (no pair) do not skip next
  2C         00EE ;RET   -- Return (VF=FF=No pair)
  2E         F065 ;GET   -- Get next byte (possible 2nd pair or 3's)

0930         3E00 ;SK=00 -- If on 2nd pass (@ 06E0-0716) skip next
  32         8F00 ;VF=V0 -- Let VF=2nd pair (highest because of sort)
  34         00EE ;RET   -- Then return (VF=pair 1, pair 2 or FF byte)
  36         0000      FILLER
  38         0000      FILLER
```

MLS - CLEAR TEXT LINE

```
093A  F8  LDI      ;Clear memory bytes 0F50-0F8F
  3B  0F
  3C  BC  PHI   RC ;Set RC=0F50, the address
  3D  F8  LDI
  3E  50           ;the display message area
  3F  AC  PLO   RC

0940  F8  LDI      ;Load D register with 00 byte
  41  00
```

```
0942  5C  STR   RC ;Store @ M(R(c)) to erase
  43  1C  INC   RC ;RC=RC+1
  44  8C  GLO   RC
  45  FB  XRI      ;Test RC.0
  46  90           ;When=90, all bytes erased
  47  3A  BNZ      ;Else branch back to do another byte
  48  40
  49  D4  SEP   R4 ;Return (also erases user's draw cards markers)
```

MLS - LOOK UP MESSAGE

```
094A  F8  LDI      ;(Chip-8 V0 passes message number)
  4B  F0
  4C  A6  PLO   R6 ;R6 points to V0
  4D  93  GHI   R3
  4E  BC  PHI   RC ;RC.1=R3.1 (page address--RC is PC)
  4F  06  LDN   R6 ;Get message # (00-0F possible)

0950  F9  ORI
  51  A0           ;OR with A0 to address table
  52  AC  PLO   RC ;RC points to look up table of messages
  53  0C  LDN   RC
  54  AA  PLO   RA ;RA.0= address of message
  55  F8  LDI
  56  0D
  57  BA  PHI   RA ;RA.1= page 0D00 on which messages exist
  58  D4  SEP   R4 ;Return
```

SET I - MLS TO SET I TO ASCII STRING FOR HAND TYPES

INPUT: RA - Eval slot for the player
OUTPUT:RA - Bit pattern for that hand

```
0959  0A  LDN   RA ;Get the evaluation  (on page 0B)
  5A  FA  ANI      ;(RA="I" set before calling sub)
  5B  0F           ;"AND" with 0F to strip first 4 bits
  5C  F9  ORI      ;"OR" with 90 to form address in table
  5D  90
  5E  AA  PLO   RA ;RA.0=address
  5F  93  GHI   R3
```

```
0960  BA  PHI   RA ;RA.1=R3.1 -- RA addresses table
  61  0A  LDN   RA ;Get byte from table
  62  AA  PLO   RA ;RA.0= RA addresses ASCII string
  63  F8  LDI
  64  08            ;RA.1= page address of ASCII strings
  65  BA  PHI   RA
  66  D4  SEP   R4 ;Return
  67  00        FILLER
```

COUNT ACTIVE PLAYERS

```
0968  COUNT :6100 ;V1=00 -- Initialize count = 00
  6A         6401 ;V4=01 -- Initialize player index = 01
  6C  CNT1   :2518 ;AMT  -- Get total/fold flag/V0=AMT
  6E         30FF ;SK=FF -- Skip if player has folded

0970         8240 ;V2=V4 -- Else mark most recent active
  72         30FF ;SK=FF -- Skip on fold (V0=FF)
  74         7101 ;V0+01 -- Count the active player
  76         7401 ;V4+01 -- Next player
  78         3405 ;SK=05 -- Skip when past last player (V4=05)
  7A         196C ;CNT1  -- Loop till done
  7C         00EE ;RET   -- Return (V1= # actives, V2= last if only one)
```

SHUFFLE BLUFF MASKS SUB

```
097E  BLFL  :6210 ;V2=10 -- V2 = loop count of 10 hex (do 16 times)
  80  BLFL1 :AC0C  MASKS -- Set "I" to bluffing masks @ 0C0C
  82         C003 ;RND   -- V0=RND # 00-03 (index #1)
  84         C103 ;RND   -- V1=RND # 00-03 (index #2)
  86         0B00 ;XCHNG -- Do MLS -- exchange two bytes @ I indexed V0 V1
  88         72FF ;V2-01 -- Subtract 01 from loop count
  8A         3200 ;SK=00 -- Skip to exit when loop count = 00
  8C         1980  BLFL1 -- Else jump back to shuffle again
  8E         00EE ;RET   -- Return -- strategies set
```

HAND TYPES LOOK-UP ADDRESS TABLE

```
0990  C6   ZILCH -- Low order address ASCII string on page 0800
  91  CC   1 PAIR--    "          "          "          "
```

```
0992  D3  2 PAIR --    "          "          "          "
  93  DA  THREES --    "          "          "          "
  94  E1   STRT  --    "          "          "          "
  95  E6  FLUSH  --    "          "          "          "
  96  EC  FULHSE --    "          "          "          "
  97  F3   FOURS --    "          "          "          "
  98  F9  STRTFL --    "          "          "          "
  99  00  (FILLER)
  9A  00     "
  9B  00     "
  9C  00     "
  9D  00     "
  9E  00     "
  9F  C0   FOLDS --    "          "          "          "
```

MESSAGES LOOK-UP ADDRESS TABLE

```
09A0  6C  SHUFFLING  -- Low order address ASCII string on page 0D00
  A1  76  YOUR HAND  --    "          "          "          "
  A2  80  PLEASE BET --    "          "          "          "
  A3  8B  BET CALLED --    "          "          "          "
  A4  96    DEALER   --    "          "          "          "
  A5  9E  BET RAISED --    "          "          "          "
  A6  A9   CARDS ?   --    "          "          "          "
  A7  B0  TRY AGAIN  --    "          "          "          "
  A8  00  (FILLER)
  A9  00     "
  AA  00     "
  AB  00     "
  AC  00     "
  AD  00     "
  AE  00     "
  AF  00     "

09B0  00     "
  B1  00     "

09B2  00  Betting total work byte (for use in betting sub)
  B3  00  Betting amount work byte (betting modules store bet here)

09B4  00  20  40  9E  40  20  00  00  -- Bit pattern arrow #1 (left)
09BC  08  14  22  08  08  08  08  08  -- Bit pattern arrow #2 (up)
09C4  00  04  02  79  02  04  00  00  -- Bit pattern arrow #3 (right)
09CC  10  10  10  10  10  44  28  10  -- Bit pattern arrow #4 (down)
09D4  42  45  54  00                  -- ASCII string for "BET"
09D8  52  53  45  00                  -- ASCII string for "RSE"
09DC  50  4F  54  00                  -- ASCII string for "POT"
```

PLAYERS' CASH STORAGE BYTES

```
09E0  00  Player #1 cash -- These initialized to 32 hex
  E1  00  Player #2 cash -- by program (=$50 decimal)
  E2  00  Player #3 cash -- "          "          "
  E3  00  Player #4 cash -- "          "          "

09E4  54 45 52 52 59 40 00 -- ASCII string - name #1 (Terry)
09EB  52 49 43 4B 40 40 00 -- ASCII string - name #2 (Rick)
09F2  54 4F 4D 40 40 40 00 -- ASCII string - name #3 (Tom)
09F9  56 49 50 45 52 53 00 -- ASCII string - name #4 (Vipers)
```

NOTE: Insert your own name (in ASCII) at 09F9-09FE.
Important -- 09FF must be a 00 (null) byte!

MLS - HAND EVAL - COUNT PAIRS

```
0A00  22  DEC   R2 ;Stack free
  01  9A  GHI   RA
  02  B7  PHI   R7 ;R7.1 = RA.1
  03  BC  PHI   RC ;RC.1 = RA.1
  04  8A  GLO   RA
  05  A7  PLO   R7 ;R7.0 = RA.0 (R7=RA)
  06  FC  ADI      ;Add 06 hex to RA.0
  07  06
  08  AC  PLO   RC ;RC.0=result - pointer to pairs in eval
  09  AF  PLO   RF ;RF.0=result - save the pairs address
  0A  8A  GLO   RA ;D = RA.0
  0B  FC  ADI      ;D = D+04
  0C  04
  0D  BF  PHI   RF ;RF.1=D (RF.1=last card address)
  0E  F8  LDI                ;Begin count
  0F  00

0A10  AE  PLO   RE ;RE.0=0
  11  17  INC   R7 ;R7.0=R7.0+1
  12  0A  LDN   RA ;D=M(R(A))  Get card @ N
  13  52  STR   R2 ;Push for comparing
  14  07  LDN   R7 ;D=M(R(7))  Get card @ N+i
  15  F3  XOR      ;Compare N:N+i
  16  FA  ANI      ;"AND" with 0F to strip suit information
  17  0F
  18  3A  BNZ      ;Branch if ≠ to 0A21
  19  21
  1A  1E  INC   RE ;Else RE.0=RE.0+1  (count the match)
```

```
0A1B  87  GLO   R7 ;D=R7.0 (current i)
  1C  52  STR   R2 ;Push for comparing
  1D  9F  GHI   RF ;D=RF.1 (last card address)
  1E  F3  XOR      ;Compare R7.0:RF.1
  1F  3A  BNZ      ;If ≠, branch to 0A11

0A20  11
  21  8E  GLO   RE
  22  32  BZ       ;Branch if =0 to 0A31 (no matches)
  23  31
  24  FC  ADI      ;Else add 1 to value
  25  01
  26  FE  SHL      ;Else move count to high 4 bits
  27  FE  SHL      ;by shifting left four times
  28  FE  SHL
  29  FE  SHL
  2A  52  STR   R2 ;Push for "OR"ing
  2B  0A  LDN   RA ;D=M(R(A)) Get card N (which has matches)
  2C  FA  ANI      ;"AND" with 0F to strip suit
  2D  0F
  2E  F1  OR       ;"OR" with count on stack (byte packed)
  2F  5C  STR   RC ;M(R(C))=D  Store in pairs area

0A30  1C  INC   RC ;RC=RC+1
  31  87  GLO   R7
  32  AA  PLO   RA ;RA.0=R7.0  (advance to next unequal card)
  33  52  STR   R2 ;but push value for comparing
  34  9F  GHI   RF ;D=RF.1 (get last card address)
  35  F3  XOR      ;Compare RA.0:RF.1
  36  3A  BNZ      ;If ≠, branch to 0A0E  (not done)
  37  0E
  38  F8  LDI      ;Else store stop byte after pairs (or no pairs)
  39  FF
  3A  5C  STR   RC ;M(R(C))=FF
  3B  1C  INC   RC ;RC=RC+1  (RC=TA now)

                      (RECOMMEND THROW OUTS)

0A3C  9A  GHI   RA ;D=RA.1  (hand high order address)
  3D  BD  PHI   RD ;RD.1=D  (RD = pairs pointer for this test)
  3E  9F  GHI   RF ;D=RF.1  (get address of last card)
  3F  FF  SMI      ;D=D-04  (subtract 04)

0A40  04
  41  AA  PLO   RA ;RA.0=D  (reset RA to first card)
  42  8F  GLO   RF ;D=RF.0  (get saved pairs address)
  43  AD  PLO   RD ;RD.0=D  (RD = pairs pointer)
  44  0D  LDN   RD ;D=M(R(D))  (get a possible pair value)
  45  FB  XRI      ;           (but test if = FF  stop byte)
```

```
0A46  FF
  47  32  BZ       ;If = FF, branch to 0A5D
  48  5D
  49  4D  LDA  RD ;D=M(R(D))  (get the definite pair value)
  4A  52  STR  R2 ;Push for comparing
  4B  0A  LDN  RA ;D=M(R(A))  (get a card)
  4C  F3  XOR      ;Compare with byte on stack
  4D  FA  ANI      ;"AND" with 0F to strip undefined first 4 bits
  4E  0F
  4F  3A  BNZ      ;If ≠, branch to 0A44  (no match)

0A50  44
  51  9F  GHI  RF ;D=RF.1  (last card address)
  52  52  STR  R2 ;Push
  53  8A  GLO  RA ;D=RA.0
  54  1A  INC  RA ;RA=RA+1
  55  F3  XOR      ;Compare RF.1:RA.0
  56  3A  BNZ      ;If ≠, branch to 0A42  (continue)
  57  42
  58  F8  LDI
  59  FF
  5A  5C  STR  RC ;M(R(C))=FF  (store FF stop byte after throw outs)
  5B  30  BR       ;Branch to next section
  5C  62
  5D  0A  LDN  RA ;D=M(R(A))
  5E  5C  STR  RC ;M(R(C))=D  (store throw away)
  5F  1C  INC  RC ;RC=RC+1

0A60  30  BR       ;Branch to 0A51
  61  51

                     (TYPE DECODING PART 1)

0A62  8F  GLO  RF
  63  AD  PLO  RD ;RD.0=RF.0  (RD points to possible pairs)
  64  0D  LDN  RD ;D=M(R(D))  (get possible pair)
  65  FB  XRI      ;Compare with FF
  66  FF
  67  32  BZ       ;If=FF, branch to 0A83  (go test straights/flushes)
  68  83                                    (there are no pairs)
  69  4D  LDA  RD ;Else D=M(R(D)); RD=RD+1 (get definite pair)
  6A  F6  SHR      ;Shift right x 4 to get only the pair's number
  6B  F6  SHR      ;    "                "                "
  6C  F6  SHR      ;    "                "                "
  6D  F6  SHR      ;    "                "                "
  6E  AE  PLO  RE ;RE.0=D   (store result--≠0--in RE.0 = J)
  6F  0D  LDN  RD ;D=M(R(D)) (get next possible pair)
```

```
0A70  FB  XRI      ;Compare with FF
  71  FF
  72  32  BZ       ;If=FF, branch to 0A79  (there is only one pair)
  73  79
  74  0D  LDN  RD  ;D=M(R(D))  (get same definite pair)
  75  F6  SHR      ;Shift right x 4 to get only the pair's number
  76  F6  SHR      ;    "                 "                "
  77  F6  SHR      ;    "                 "                "
  78  F6  SHR      ;    "                 "                "
  79  52  STR  R2  ;Push  (J1)  (either a pair # or a zero byte)
  7A  8E  GLO  RE  ;D=RE.0      (get J value)
  7B  F5  SD       ;Subtract M(R(X))-D  (J1-J)
  7C  33  BPZ      ;If J1≥J then branch to 0A80  (D holds correct value)
  7D  80
  7E  8E  GLO  RE  ;D=RE.0  else get same value of J
  7F  F7  SM       ;Subtract  D-M(R(X))  subtract other way around

0A80  BE  PHI  RE  ;RE.1=D   (save the embryo value)
  81  30  BR       ;Branch to 0AC4  (final decode)
  82  C4

                        (STRAIGHTS)

0A83  F8  LDI
  84  00
  85  BE  PHI  RE  ;RE.1=0  (to initialize)
  86  9F  GHI  RF  ;D=RF.1  (get last card address)
  87  FF  SMI      ;Subtract 04
  88  04
  89  AA  PLO  RA  ;RA.0=D  (RA points to first card)
  8A  0A  LDN  RA  ;D=M(R(A))  (get first card)
  8B  FA  ANI      ;"AND" with 0F to strip suit
  8C  0F
  8D  52  STR  R2  ;Push for sequential comparison
  8E  F0  LDX      ;Pop value off stack
  8F  FC  ADI      ;Add 01

0A90  01
  91  52  STR  R2  ;Push new value
  92  1A  INC  RA  ;RA=RA+1  (next card)
  93  0A  LDN  RA  ;D=M(R(A))  (get next card)
  94  FA  ANI      ;"AND" with 0F to strip suit
  95  0F
  96  F3  XOR      ;Compare with byte on stack
  97  3A  BNZ      ;If ≠, then not a straight, branch to 0AA4
  98  A4                    (go test flush)
  99  22  DEC  R2  ;Decrement stack to preserve value
  9A  8A  GLO  RA  ;D=RA.0
  9B  52  STR  R2  ;Push for comparing
```

```
0A9C  9F  GHI   RF ;D=RF.1
  9D  F3  XOR      ;Compare RA.0:RF.1
  9E  12  INC   R2 ;Reset stack pointer
  9F  3A  BNZ      ;If ≠, branch to 0A8E

0AA0  8E
  A1  F8  LDI      ;Else hand is a straight
  A2  05
  A3  BE  PHI   RE ;RE.1=05 to mark straight

                          (FLUSHES)

0AA4  F8  LDI
  A5  07
  A6  AE  PLO   RE ;RE.0=07  (initialize RE.0)
  A7  9F  GHI   RF ;D=RF.1
  A8  FF  SMI      ;Subtract 04
  A9  04
  AA  AA  PLO   RA ;RA.0=D   (RA points to first card)
  AB  0A  LDN   RA ;D=M(R(A))  (get card)
  AC  52  STR   R2 ;Push
  AD  1A  INC   RA ;RA=RA+1  (next card)
  AE  0A  LDN   RA ;D=M(R(A))  (get card)
  AF  F3  XOR      ;Compare with byte on stack

0AB0  FA  ANI      ;"AND" result with F0 to strip card type
  B1  F0
  B2  3A  BNZ      ;If ≠, branch to 0ABF  (no flush)
  B3  BF
  B4  22  DEC   R2 ;Decrement stack pointer preserves value
  B5  8A  GLO   RA ;D=RA.0
  B6  52  STR   R2 ;Push for comparing
  B7  9F  GHI   RF ;D=RF.1
  B8  F3  XOR      ;Compare RA.0:RF.1
  B9  12  INC   R2 ;Reset stack pointer
  BA  3A  BNZ      ;If RA.0≠RF.1, branch to 0AAD to continue
  BB  AD
  BC  F8  LDI      ;Else hand is a flush (at least)
  BD  06
  BE  AE  PLO   RE ;RE.0=06  (to mark flush)
  BF  8E  GLO   RE ;D=RE.0  (either 06 or 00)

0AC0  52  STR   R2 ;Push for adding
  C1  9E  GHI   RE ;D=RE.1  (either 05 or 00 -- straight or no straight)
  C2  F4  ADD      ;D=RE.1 + RE.0
  C3  BE  PHI   RE ;RE.1=D  (RE.1=05=straight/06=flush/0B=straight
                             flush/00=bust hand)
```

(FINAL DECODE)

```
0AC4  8F  GLO   RF ;D=RF.0
  C5  AD  PLO   RD ;RD.0=D  (RD points to first possible pair)
  C6  2D  DEC   RD ;RD=RD-1 (RD points to hand eval slot)
  C7  93  GHI   R3 ;D=PC.1  (get this page value)
  C8  BF  PHI   RF ;RF.1=D  (put in RF.1)
  C9  F8  LDI
  CA  EC
  CB  AF  PLO   RF ;RF.0=EC (RF=top of look-up table @ 0AEC)
  CC  9E  GHI   RE ;D=RE.1  (get embryo hand value)
  CD  52  STR   R2 ;Push for comparing with table entries
  CE  38  SKP      ;Skip next - first time through only
  CF  1F  INC   RF ;RF=RF+1 - next table entry

0AD0  4F  LDA   RF ;D=M(R(F)); RF=RF+1 (points to byte after key)
  D1  F3  XOR      ;Compare RE.1:M(R(F))
  D2  3A  BNZ      ;If ≠, branch to 0ACF
  D3  CF
  D4  0F  LDN   RF ;D=M(R(F))  (get matched table entry)
  D5  5D  STR   RD ;M(R(D))=D  (store as hand evaluation)
  D6  0D  LDN   RD ;D=M(R(D))  (get hand evaluation)
  D7  FB  XRI      ;Compare with 04  (straight?)
  D8  04
  D9  32  BZ       ;If =, branch to 0AE5 -- cancel recommended
  DA  E5                                    throw-outs
  DB  0D  LDN   RD ;D=M(R(D))
  DC  FB  XRI      ;Compare with 05  (flush?)
  DD  05
  DE  32  BZ       ;If =, branch to 0AE5
  DF  E5

0AE0  0D  LDN   RD ;D=M(R(D))
  E1  FB  XRI      ;Compare with 08  (straight flush?)
  E2  08
  E3  3A  BNZ      ;If ≠, branch to 0AEA
  E4  EA
  E5  1D  INC   RD ;RD=RD+2  (point to TA area)
  E6  1D  INC   RD
  E7  F8  LDI
  E8  FF
  E9  5D  STR   RD ;M(R(D))=FF Stop byte (do not take any cards)
  EA  12  INC   R2 ;Reset stack pointer
  EB  D4  SEP   R4 ;Return to Chip-8 program
```

LOOK-UP CONVERSION TABLE

```
0AEC  05  FF   RE.1=05  -- FOLD            =FF -- The numbers
  EE  00  02   RE.1=00  -- TWO PAIR        =02 -- on the right

0AF0  01  06   RE.1=01  -- FULL HOUSE      =06 -- put the
  F2  02  01   RE.1=02  -- ONE PAIR        =01 -- evaluations
  F4  03  03   RE.1=03  -- 3 OF A KIND     =03 -- in their
  F6  04  07   RE.1=04  -- 4 OF A KIND     =07 -- proper
  F8  06  05   RE.1=06  -- FLUSH           =05 -- poker order
  FA  07  00   RE.1=07  -- BUST HAND       =00 --
  FC  0B  08   RE.1=0B  -- STRAIGHT FLUSH  =08 --
  FE  0C  04   RE.1=0C  -- STRAIGHT        =04
```

MLS - XCHNG - (TO SHUFFLE DECK AND STRATEGIES)

```
0B00  22  DEC   R2 ;Decrement stack pointer to free location
  01  9A  GHI   RA ;Get RA.1 (high part of "I" address)
  02  BE  PHI   RE ;Put in RE.1 - pointer A
  03  BF  PHI   RF ;Put in RF.1 - pointer B
  04  8A  GLO   RA ;Get RA.0  (low part of "I" address)
  05  52  STR   R2 ;Push onto stack
  06  F8  LDI      ;Load "F0" byte into D register
  07  F0
  08  A6  PLO   R6 ;Put in R6.0 to address Chip-8's V0 variable
  09  46  LDA   R6 ;Get value of V0, advance R6 to V1
  0A  F4  ADD      ;Add to byte on stack forming random index
  0B  AE  PLO   RE ;Put in RE.0 - pointer A
  0C  06  LDN   R6 ;Get value of V1
  0D  F4  ADD      ;Add to byte on stack forming random index
  0E  AF  PLO   RF ;Put in RF.0 - pointer B
  0F  0E  LDN   RE ;Get byte (card #1) at pointer A

0B10  52  STR   R2 ;Push onto stack to store temporarily
  11  0F  LDN   RF ;Get byte (card #2) at pointer B
  12  5E  STR   RE ;Store card #2 at old card #1 position
  13  72  LDXA     ;Pop card #1 resetting stack pointer
  14  5F  STR   RF ;Store card #1 at old card #2 position
  15  D4  SEP   R4 ;Return control to Chip-8 Interpreter
```

MLS - SORT HAND

```
0B16  22  DEC   R2 ;Stack free
  17  94  GHI   R4 ;(D=00); Begin sort
  18  AD  PLO   RD
  19  9A  GHI   RA ;RE=RA = i = data base
  1A  BE  PHI   RE
  1B  8A  GLO   RA
  1C  AE  PLO   RE
  1D  8D  GLO   RD ;Get # "good" comparisons
  1E  FB  XRI
  1F  04        ;Test if = N-1 (N=5 cards in each hand)

0B20  3A  BNZ
  21  24
  22  12  INC   R2 ;Inc stack pointer to reset
  23  D4  SEP   R4 ;Return control to Chip-8 Interpreter
  24  1D  INC   RD ;Count the comparison
  25  4E  LDA   RE ;Get byte Xi - (i+1)
  26  BF  PHI   RF ;RF.1 holds Xi
  27  FA  ANI       ;"AND" with 0F to
  28  0F            ;strip off suit
  29  52  STR   R2 ;Push for comparing
  2A  0E  LDN   RE ;Get byte Xi+1
  2B  AF  PLO   RF ;RF.0 holds Xi+1
  2C  FA  ANI       ;"AND" with 0F to
  2D  0F            ;strip off suit
  2E  F7  SM        ;Subtract  D-M(R(X)) ((Xi+1)-Xi)
  2F  33  BPZ       ;If (Xi+1)≥ Xi, branch to 0B1D

0B30  1D              no need to sort - order correct
  31  9F  GHI   RF ;Else get Xi      S         B
  32  5E  STR   RE ;Store @ i+1       W         Y
  33  2E  DEC   RE ;i-1                 A         T
  34  8F  GLO   RF ;Get  Xi+1            P         E
  35  5E  STR   RE ;Store @  i                      S
  36  30  BR        ;Loop for next set of comparisons
  37  17              to 0B17
```

MLS - DISPLAY HANDS DECODING

```
0B38  8A  GLO   RA
  39  AF  PLO   RF ;RF.0=RA.0 (save address top card in hand)
  3A  F8  LDI
  3B  0C
```

```
0B3C  BC  PHI   RC ;RC.1=0C  (page address work area)
  3D  F8  LDI
  3E  24
  3F  AC  PLO   RC ;RC=21 byte work area @ 0C24

0B40  F8  LDI
  41  05
  42  AD  PLO   RD ;RD.0=05 loop count
  43  4A  LDA   RA ;D=M(R(A))  (get card)
  44  FA  ANI      ;Strip suit by "AND"ing with 0F byte
  45  0F
  46  F9  ORI
  47  30           ;Combine with 30 for ASCII 3N via "OR" function
  48  5C  STR   RC ;Put in work area
  49  1C  INC   RC ;Advance pointer in work area
  4A  F8  LDI
  4B  20
  4C  5C  STR   RC ;Store an ASCII space (20 hex)
  4D  1C  INC   RC ;Advance pointer in work area
  4E  2D  DEC   RD ;Loop - 01
  4F  8D  GLO   RD

0B50  3A  BNZ
  51  43           ;Loop till all 5 card types done to 0B43
  52  5C  STR   RC ;M(R(C))=00 (null byte needed by messager)
  53  1C  INC   RC ;Advance pointer in work area

                        (SUITS)

0B54  F8  LDI
  55  05
  56  AD  PLO   RD ;RD=05  Loop count
  57  8F  GLO   RF
  58  AA  PLO   RA ;RA.0=RF.0 to reset to top card in hand
  59  4A  LDA   RA ;D=M(R(A))  (get card)
  5A  F6  SHR      ;Shift right x 4 for suit only  (four MSB's)
  5B  F6  SHR      ;   "      "     "      "
  5C  F6  SHR      ;   "      "     "      "
  5D  F6  SHR      ;   "      "     "      "
  5E  5C  STR   RC ;Store in work area (ASCII conversion="ON" byte)
  5F  1C  INC   RC ;Advance pointer in work area

0B60  F8  LDI
  61  20
  62  5C  STR   RC ;Store an ASCII space (20 hex)
  63  1C  INC   RC ;Advance pointer in work area
  64  2D  DEC   RD ;Loop count - 01
  65  8D  GLO   RD
  66  3A  BNZ      ;If loop count ≠ 00 yet, branch to 0B59
  67  59               (decode 5 card suits)
  68  5C  STR   RC ;M(R(C))=00 (null stop byte for messager)
  69  D4  SEP   R4 ;Return control to Chip-8 Interpreter
```

MLS - LET V1 V2 V3 V4= EVALUATIONS (+1)

```
0B6A  F8  LDI
  6B  F1
  6C  A6  PLO   R6 ;R6.0=F1; points to Chip-8 V1 variable
  6D  0A  LDN   RA ;D=M(R(A)); get an evaluation (RA=I preset by caller)
  6E  FC  ADI      ;Add 01
  6F  01

0B70  56  STR   R6 ;M(R(6))=D; store as V1 V2 V3 or V4
  71  16  INC   R6 ;R6+1  point to next Chip-8 variable
  72  8A  GLO   RA
  73  FC  ADI      ;Add 0F to RA.0 for next evaluation
  74  0F
  75  AA  PLO   RA ;RA.0=D
  76  FB  XRI      ;But test if RA past last evaluation
  77  D1
  78  3A  BNZ      ;If not, branch to 0B6D to continue
  79  6D
  7A  D4           ;Else return control to Chip-8 Interpreter
```

MLS - ANTE

```
0B7B  F8  LDI
  7C  FB
  7D  A6  PLO   R6 ;R6.0=FB  R6 points to Chip-8's VB (POT)
  7E  06  LDN   R6 ;D=M(R(6))  Get pot value
  7F  AC  PLO   RC ;RC.0 holds pot value

0B80  0A  LDN   RA ;Get player's cash  (RA="I" preset by caller)
  81  32  BZ       ;If = 00, branch to Do next player to 0B87
  82  87
  83  FF  SMI      ;Else subtract 01 for ante from cash
  84  01
  85  5A  STR   RA ;And replace @ M(R(A))
  86  1C  INC   RC ;RC+1 to add $1 to pot for each ante
  87  8A  GLO   RA ;D=RA.0 to see if done yet
  88  1A  INC   RA ;RA=RA+1  for next cash amount
  89  FB  XRI      ;Test if RA.0=E3  (last cash amount)
  8A  E3
  8B  3A  BNZ      ;If not, branch up to Do next to 0B80
  8C  80
  8D  8C  GLO   RC ;Get new pot value
  8E  56  STR   R6 ;Replace as Chip-8 VB variable (+ antes)
  8F  D4  SEP   R4 ;Return control to Chip-8 Interpreter
```

HANDS & EVALUATIONS

```
0B90   5 empty bytes -- Hand #1  (Terry)
0B95  10    "     "  -- Evaluation #1
0B9F   5    "     "  -- Hand #2  (Rick)
0BA4  10    "     "  -- Evaluation #2
0BAE   5    "     "  -- Hand #3  (Tom)
0BB3  10    "     "  -- Evaluation #3
0BBD   5    "     "  -- Hand #4 (User--"VIPERS")
0BC2  10    "     "  -- Evaluation #4
```

DECK OF CARDS

```
0BCC  12  13  14  15  16  17  18  19  1A  1B  1C  1D  1E  (Hearts)
0BD9  22  23  24  25  26  27  28  29  2A  2B  2C  2D  2E  (Clubs)
0BE6  42  43  44  45  46  47  48  49  4A  4B  4C  4D  4E  (Diamonds)
0BF3  82  83  84  85  86  87  88  89  8A  8B  8C  8D  8E  (Spades)
```

0C00-0DFF 2 Pages -- ASCII character set -- (format described in the author's book, PIPS for VIPS)
The character set is modified to include card suits and additional letters in place of some punctuation.

The subroutines and data areas below are interspersed within the character set replacing bit patterns for unused characters. Areas that do contain character bit patterns are labled -- "more ASCII characters"

```
0C00   (4 bytes)     -- Data area for storing # cards drawn
0C04   (8 bytes)     -- More ASCII characters
0C0C  7F 0F 0F 03    -- Bluffing strategies (may be any CXKK masks)
0C10   (4 bytes)     -- More ASCII characters
```

MLS - SET HAND = FF's (TO FOLD PLAYER)

```
0C14  F8  LDI
  15  05
```

```
0C16  AC  PLO   RC ;RC.0=05 = Loop count
  17  F8  LDI
  18  FF
  19  5A  STR   RA ;Store FF byte in hand
  1A  1A  INC   RA ;RA=RA.1 next card
  1B  2C  DEC   RC ;RC=RC.1  Count loop
  1C  8C  GLO   RC ;Test loop
  1D  3A  BNZ      ;If ≠ 00, loop back to continue to 0C17
  1E  17
  1F  D4  SEP   R4 ;Else return control to Chip-8 Interpreter


0C20    (4 bytes)    -- More ASCII characters
0C24    (22 bytes)   -- Work area for ASCII hand decoding (21 bytes used)
0C3A      2810       -- Bit pattern for user draw cards marker
0C3C    (4 bytes)    -- More ASCII characters


                  MLS - DRAW CARDS


0C40  22  DEC   R2 ;Stack pointer free
  41  F8  LDI
  42  FE
  43  A6  PLO   R6 ;R6.0=FE (points to Chip-8 VE)
  44  8A  GLO   RA ;D=RA.0
  45  AF  PLO   RF ;RF.0=RA.0; save hand address in RF.0
  46  FC  ADI
  47  04
  48  BF  PHI   RF ;RF.1=RA.0+4; Figure last card address in RF.1
  49  AC  PLO   RC
  4A  1C  INC   RC ;RC.0=RA.0+5; point to eval
  4B  9A  GHI   RA
  4C  BC  PHI   RC ;RC.1=RA.1; RC now set (no carry possible)
  4D  4C  LDA   RC ;Get byte in eval
  4E  FB  XRI
  4F  FF           ;Look for FF stop byte

0C50  3A  BNZ
  51  4D           ;Continue till FF found (RC points to throw-outs)
  52  8C  GLO   RC
  53  FC  ADI
  54  03
  55  AA  PLO   RA ;Set RA= the end of 3 cards maximum for draw
  56  F8  LDI          (in case EVAL recommends more than 3)
  57  FF
  58  5A  STR   RA ;Store an FF stop byte there
```

```
0C59  9A  GHI   RA
  5A  BD  PHI   RD ;RD.1=RA.1
  5B  06  LDN   R6 ;Get next free address in deck (VE)
  5C  AD  PLO   RD ;RD.0=M(R(6))=VE
  5D  8F  GLO   RF ;D=RF.0; Begin
  5E  AA  PLO   RA ;RA.0=D  Restore RA.0 to first card
  5F  4C  LDA   RC ;D=M(R(C))  Get a throw away

0C60  52  STR   R2 ;Push for comparing to hand
  61  FB  XRI      ;But check if = FF stop byte
  62  FF
  63  3A  BNZ      ;If ≠ FF, branch to 0C69
  64  69
  65  8D  GLO   RD ;(Begin exit)
  66  56  STR   R6 ;M(R(6)) (VE)=RD.0 to save index
  67  12  INC   R2 ;Reset stack pointer
  68  D4  SEP   R4 ;Return control to Chip-8 Interpreter
  69  0A  LDN   RA ;Get a card in hand
  6A  F3  XOR      ;Compare with card on stack
  6B  3A  BNZ      ;If ≠, skip next exchange - continue
  6C  71                                  to search
  6D  4D  LDA   RD ;Get card in deck
  6E  5A  STR   RA ;Store in hand
  6F  30  BR       ;Branch to 0C5D for another search

0C70  5D
  71  1A  INC   RA ;Else RA=RA+1 to continue search
  72  30  BR       ;Branch to 0C69
  73  69


0C74      (12 bytes)    -- Available -- not used by program
0C80      (12 bytes)    -- More ASCII characters
```

BET MODULE CONTROL SUB

```
0C8C  MOD+  :6007 ;V0=07 -- Message #7 (TRY AGAIN)
  8E         283E ;MCENT -- Do sub -- display error message

0C90  MOD   :3404 ;SK=04 -- Skip next only for user (player #4)
  92         138A  BET   -- Go betting modules other players
  94         6100 ;V1=00 -- Initialize V1=00=answer
  96  MOD1   :F00A ;V0=KY -- Get keypress in V0 (waits)
  98         400E ;SK≠0E -- If ≠ 0E, skip to continue
  9A         1CA2 ;MOD2  -- Else go to exit
```

```
0C9C          01A4 ;MLS    -- Do MLS -- combine keypress in answer
  9E          F1F0 ;V1=V0  -- VX VY passed to MLS in interpreter

0CA0          1C96 ;MOD1   -- Go loop for next instruction
  A2  MOD2    410F ;SK≠0F  -- Skip if player is not folding
  A4          15B4 ;FOLD   -- Go fold player & return
  A6          2CB8 ;>CASH  -- Do sub -- check cash vs. amt bet
  A8          3F01 ;SK=01  -- If VF=00, Skip; player has enough cash
  AA          1C8C ;MOD+   -- Else go restart
  AC          289E ;LEGAL  -- Do sub to check if bet is legal
  AE          3F01 ;SK=01  -- If VF=00, skip; bet is legal

0CB0          1C8C ;MOD+   -- Else go restart
  B2          15E4 ;STRBT  -- Go place bet @ 09B3 (00EE return there)


0CB4   (4 bytes)   -- More ASCII characters
```

BET GREATER THAN CASH SUB

```
0CB8  BET>$ :2524 ;CSHIN  -- Do sub -- set "I" to player's (V4=#) cash
  BA          F065 ;GET    -- Let V0=cash for this player
  BC          8015 ;V0-V1  -- Subtract amount bet from cash
  BE          00EE ;RET    -- Return (VF=00=bet>cash/VF=01=bet≤cash)


0CC0   (87 bytes)  -- More ASCII characters
```

ASCII MESSAGE STRINGS

```
0D6C  53 48 55 46 46 4C 49 4E  (0D6C="SHUFFLING")
0D74  47 00 59 4F 55 52 20 48  (0D76="YOUR HAND")
0D7C  41 4E 44 00 50 4C 45 41  (0D80="PLEASE BET")
0D84  53 45 20 42 45 54 00 42  (0D8B="BET CALLED")
0D8C  45 54 20 43 41 4C 4C 45
0D94  44 00 44 45 41 4C 45 52  (0D96="DEALER")
0D9C  00 00 42 45 54 20 52 41  (0D9E="BET RAISED")
0DA4  49 53 45 44 00 43 41 52  (0DA9="CARDS ?")
0DAC  44 53 3F 00 54 52 59 20  (0DB0="TRY AGAIN")
0DB4  41 47 41 49 4E 00
```

FIGURE BET (PART OF MODULE BET SUB)

```
0DBA  FIGR  ;4A00 ;SK≠00 -- If raise ≠ 00, skip next
  BC         7203 ;V2+03 -- Weight + 3 (encourage opening, staying)
  BE         8190 ;V1=V9 -- Let V1 = minimum bet in V9

0DC0         6082 ;V0=82 -- Let V0 = fold threshold of 82
  C2         8025 ;V0-V2 -- Subtract fold threshold - weight
  C4         3F01 ;SK=+  -- If positive, skip next (weight≤82)
  C6         1DCE  FIGR1 -- Go continue
  C8         3900 ;SK=00 -- If bet=00 (no openers yet) skip next
  CA         15B4  FOLD  -- Go fold on hands weighted≤82)
  CC         15E4  STRBT -- Go bet $0 to pass opening
  CE  FIGR1 ;6087 ;V0=87 -- Let V0 = raise threshold of 87

0DD0         8025 ;V0-V2 -- Subtract raise threshold - weight
  D2         3F00 ;SK=-  -- If negative, skip next (weight>87)
  D4         15E4  STRBT -- Go bet minimum (in V1)
  D6  RAISE ;4A03 ;SK=03 -- Skip into raise routine if< 3 raises
  D8         15E4  STRBT -- Go bet minimum  (3 raises maximum)
  DA         6094 ;V0=94 -- Let V0=high hand threshold of 94
  DC         8025 ;V0-V2 -- Subtract threshold - weight
  DE         3F01 ;SK=+  -- If positive, skip next (weight≤94)

0DE0         72F4 ;V2-0C -- Weight - 0C (bluffs high hands/i.e. straights)
  E2         7103 ;V1+03 -- Minimum + 3 (raise or open $3)
  E4         608E ;V0=8E -- Let V0= raise $3 threshold
  E6         8025 ;V0-V2 -- Subtract threshold - weight
  E8         3F00 ;SK=00 -- If negative, skip next (weight>8E)
  EA         71FF ;V1-01 -- Bet - 01 (raise or open $2)
  EC         608A ;V0=8A -- Let V0 = raise $2 threshold
  EE         8025 ;V0-V2 -- Subtract threshold - weight

0DF0         3F00 ;SK=00 -- If negative, skip next (weight>8A)
  F2         71FF ;V1-01 -- Bet - 01 (raise or open $1)
  F4  ADJST ;2CB8  BET>$ -- Do sub -- check player's cash level
  F6         3F00 ;SK=00 -- If VF=00, then raise/open is>cash
  F8         15E4  STRBT -- Go place bet in V1 (00EE in next part)
  FA         71FF ;V1-1  -- Subtract $1 from bet
  FC         1DF4  ADJST -- Loop back to try again
  FE         0000       (FILLER)
```

V I P - F L O P

by

Tom Swan

INSTRUCTIONS

Othello*, the box game similar to your Cosmac VIP-FLOP, has been a popular game on computers for several reasons. For one, it is not terribly difficult to program though the method of figuring the computer's moves duplicates chess, checkers and other games using a look-ahead feature to make up the computer's mind. Also, the complexity of manipulating the pieces during play -- there is a good chance that players will make errors in flopping poker chips on a board -- make this game an excellent choice for a television display. The computer handles all these chores so the board is always "right," something a box game cannot do.

VIP-FLOP is supplied with three versions. You

*Registered Trademark of CBS Corporation

can play against the computer or with a friend. The computer will even play itself, a feature that can be used to demonstrate the game to a beginner or as a way to test new evaluation routines that you may write and insert. (I'll show you how in a minute.) Nine levels of play are possible and the computer will require from one to two seconds to as long as 15 minutes to figure its next move! At the highest level, the program looks eight moves ahead and is a very tough customer to beat. When you play with your Cosmac, you may let the computer go first, and if you are stumped during any part of the game, you may ask the computer to recommend a move for you!

When you flip the run switch up after loading 14 (E) pages from tape, the computer will soon ask you to enter a number from 0-F. Pressing Key 0 will select the lowest level of difficulty. About 2 seconds are needed for VIP-FLOP to figure its next move at this level -- a speed achieved by extensive use of machine language subroutines for the look-ahead. While level 0 is not difficult to beat, not one of the unsuspecting victims I've coaxed as guinea pigs to my hex pad were able to beat it the first time. If you are new to the game as were most of my subjects, level 0 will provide a good introduction to VIP-FLOP. As your play progresses,

you may select higher levels of play for a more difficult opponent. Hints on improving play past this point will also be given. You won't soon outgrow VIP-FLOP!

Each higher level of play causes the program to perform a deeper look-ahead to figure its next move. Pressing even-numbered keys (except Key 0) will select the same level as the next higher odd number. (Key 2 will be treated the same as Key 3, Key 4 equals Key 5, etc.) Each higher level results in a search one move deeper than the last level. Press Key 1 for a one move look ahead, for example, Key 3 for 2 moves, Key 5 for 3 moves on up to a maximum of Key F for an 8 move look ahead. It is possible to modify the program so it looks ahead all the way to the end of the game on each search, but in that case, you may want to lay in a stock of novels to read while the computer makes up its mind! Such a depth (to a maximum of 30 moves) would take hours!

After selecting the level of play you wish to compete against (or the level you want the computer to compete with itself) VIP-FLOP asks you to select one of three versions. Key 1 selects the normal Human vs. Computer (H-C) version; Key 2 calls a Computer vs. Computer (C-C) game, and Key 3 will cause VIP-FLOP to operate only as a monitor, keeping score and making your moves as you play with a friend -- the Human vs.

Human (H-H) mode.

As the computer prompts you every step of the way using the Messager addition to Chip-8 from my book Pips for Vips, you will seldom if ever have to refer to the instructions here again. Now, your computer tells you what to do next!

With these beginning chores out of the way, you are ready to play your first game of VIP-FLOP. To enter a move press the keys corresponding to the X:Y coordinate on the 8 x 8 checkerboard displayed on your TV screen. (These are similar to the XY coordinates you program your Cosmac display with.) Always enter the X, or horizontal number first. The square in the upper left is 1:1; on the bottom right, 8:8. If you make an error or select an illegal move, the computer tells you with a long error tone, and you must then enter two more numbers.

Provided the move you want is legal (it is adjacent to at least one other piece) VIP-FLOP shimmers your piece on the chosen square until you press Key E to either Enter the move, or Key F to Cancel it and change your mind. The X:Y coordinates are also displayed in the upper left corner above the score.

The computer (if you selected version #1) will then figure out its next move, signaling you when ready with

a quick series of beeps. The computer's choice is blinked first (to give you a chance to see it among what will eventually be a sea of black and white pieces), the move is made and it is again your turn.

If you want the computer to make the first move, press Key C before any moves are made. After the opening move if you press Key C when it is your turn, VIP-FLOP will suggest a move for you geared to whatever level of difficulty you had previously selected. The recommended move will shimmer on the chosen square just as if you had selected it, and you then have the option, as before, to Enter or Cancel the move with Keys E or F. There is no guarantee that you'll win by using all computer suggestions, but at least the computer does not cheat -- I think.

At the end of the game, the winner is shown and a new game (or a different version or level of play) may be selected by first pressing Key 0 to restart the game, and then following the instructions printed on the screen. You never have to flip the run switch down at the end of a game no matter how many new games you want to play.

If you already know the rules to Othello, you are now ready to play VIP-FLOP. (If these are not familiar to you, please go on to the next section.) After

playing a few games, you may want to try some of the variations listed following the rules below. But I know you'll want to get started -- so, have fun! (P.S.--If your Cosmac offers to play you for a nickel a point, stall it till you sharpen your skills. You have to watch these computers; give them an inch and they'll take 4K!)

RULES FOR PLAY

When you play against the computer, you will have the white pieces and the computer the black. It does not matter which player goes first.

The game is played on a 64 (8 x 8) square checkerboard with all squares used during a game. Four pieces, two white and two black, are placed in the center of the board to start the game. (It may be helpful for you to have VIP-FLOP running on your computer by the way while you read these instructions so you can see the action while you learn the rules. Try a level 0; Version 1 (H-C) game or a level 1, Version 2 (C-C) game for best results.)

The object of the game is to end up with more pieces than your opponent. Tie games are possible but infrequent. (I have <u>never</u> seen a tie game, however, in all my testing of VIP-FLOP!)

VIP-FLOP takes its name from the action of the game when pieces are moved onto the board. Whenever you manage to trap a continuous line of opponent pieces between at least two of your own, you capture your opponent's pieces and they flop over to your color. (The same of course is true for your opponent.) The same pieces may flop back and forth several times during the game taking the advantage with them from player to player. As you become more familiar with the play, you will appreciate that strategy is infinitely more important than the number of pieces you are able to flop during your turn. With a single well-placed move, either player may come from behind to capture the lead. VIP-FLOP is not a difficult game to learn, but it is not easy to master! One good thing -- your computer will help teach you to become a better player by showing you what moves to make (if you want). Or you may select Version #3 (C-C) and watch a "pure" computer match while you learn to play a better game.

Pieces may be captured in any direction or any number of directions on a single move. Like the spokes of a wheel radiating from wherever you place your next piece, your opponent's pieces are flopped to your color along the horizontal and vertical lines as well as all four diagonals. The flopping stops when it

reaches a piece of the moving player's color. In other words, you cannot flop pieces beyond your own pieces or past spaces -- only those pieces trapped between the moving piece and the next like color piece in line will be flopped. (If this seems confusing, try watching a C-C game, Version #3, till you catch on.)

Play alternates between players until all 64 squares are filled with pieces. Whoever has the most pieces at the end of the game wins.

SIMPLE VARIATIONS TO TRY

At the highest level of play, the computer takes more and more time to figure out its next move. You may want to keep a game running while you mow the lawn or eat dinner, but if you are not there at the moment the computer returns a move, you'll miss what you had been waiting for!

An easy way around this is to change the subroutine call that blinks the computer's move before making that move to a new call that will cause the computer's move to shimmer on the chosen square just as your moves do. VIP-FLOP will still signal with a beeper that it is ready to move, but will now wait until you are present to make the move. Pressing Key E or F at this time

will enter the move for the computer and it will then be your turn again. Enter your move and you are free to go finish the lawn -- the computer will not make another move until you tell it to.

Using the ROM system monitor, enter the following change:

```
0394    2568  SHIMR -- Do sub -- display move (shimmers)
```

This feature may also be added to Computer vs. Computer games (Version #3; C-C). By making the following change, neither opponent will move until you press Key E or Key F.

```
03BA    2568 ;SHIMR -- Do sub -- display move (shimmers)
```

If you want to experiment with a super long look-ahead, you may enter the following change to increase the number of moves that the computer considers in order to make up its mind.

```
030C    60KK  ;V0=KK -- KK=# plys deep (but see next note!)
```

With this variation, you should use only odd numbers for the KK in the instruction above. (B, D, F are the

only odd hex letters.) But if you make a mistake and use an even number, the program will automatically correct it for you. The number of moves that the computer will look ahead is that number plus one divided by 2 : (KK+1)/2. VIP-FLOP is designed to always search (unless it runs up against the end of the game) a whole move forward at all times ending its search with a look at its opponent's possibilities. Further strategy modifications are discussed in the program details to allow, for instance, a two and one-half move look-ahead with interesting results.

If you want VIP-FLOP to search to the end of the game on each look-ahead, enter 60FF at 030C (the look-ahead automatically stops when it reaches the potential end of the game so a number this high may be used.) This will take a while, though!

You should consider, however, if you do increase the look-ahead, that VIP-FLOP may not actually play a better game just because it looks deeper. While this may be true up to a point, the look-ahead may actually become less effective past that point. Any look-ahead is only as good as the evaluation of board positions it uses to construct the look-ahead "tree." Believe it or not, luck on the computer's side actually has an effect, too! Some hints on writing your own evaluations will be

covered later.

As VIP-FLOP is completely subroutine driven, the mechanics exist here for you to construct checkers and other 8 x 8 grid games using VIP-FLOP and it's graphics as the skeleton for your own programming. In fact some of the routines were written to incorporate this very feature. The concept of a look-ahead search is an important one in computer programming and careful study of the program listing may help you to understand its intricacy. But whatever satisfaction you find in the program, I thank you for trying it, and wish you the best of luck in your experiments!

PROGRAM DESCRIPTION

"Welcome to the Monkey House"
--Kurt Vonnegut

When I was a boy, I climbed a neighbor's tree eager to explore the private city of its limbs, each crook a new corner to turn, every branch an alleyway to adventure. It was early evening. The summer air inside the green cover of foliage was moist and cool and sweet smelling. I was not an experienced climber, and was secretly jealous of my friends who raced as monkeys do, without fear, swinging perilously in the tangle of green and brown grasping the sap sticky bark as if they were born to it. I was alone. I would prove my worth. I would win the key to their city. All I needed was a little practice. Today, I would earn my pass to the monkey house.

But my foot got stuck. Wedged hopelessly in between two twisty uptown streets that threatened to keep me there until who knows what evil would come to claim its hour. It grew dark. My shoe fit in an ugly deep crook as though its sole had been originally carved from the depths of that dark grasp and had finally returned home to the neighborhood of its birth. And it seemed determined to stay the night. Every effort to

escape got me deeper in trouble. There was no choice but to resort to the only weapon left to me -- I yelled for help. To my embarrassment, not only did my parents respond to the call, but a fellow primate, a natural resident of the area who knew it like the back of his hand, helped me down. The rescue was not totally successful, and though I would later leave my heart in a number of beautiful cities I've been fortunate to visit, I will never recover what I gave up to that tree. I wonder still if my shoe is up there. It would serve it right if it was.

My first attempt at VIP-FLOP felt like that. I got so hopelessly stuck in the logic of the look-ahead process that pieces of me still cling to the awful paths I took to get myself in such a hopeless mess -- pieces of my psyche I will never get back, and I actually say good riddance and "adios" to them. Now. For I discovered the key to performing a look-ahead and the key may be spelled: "S-I-M-P-L-I-F-Y."

When a computer uses a look-ahead in a game to figure its next move, it is actually playing a game with itself in high speed memory in the hopes of finding the path that leads to the best move it can make. As each step of the way in the look-ahead process presents a greatly increased number of possible paths that the

game could take, all the possible paths, if drawn on a piece of paper, would resemble an upside down tree with its branches (the possible paths) sticking out of the bottom, its trunk a single line reaching up to the sky.

The trunk of the tree begins at a single point representing one of all the possible moves the computer will consider to figure the best (hopefully) move of all those possible. At each "node" in the tree, those dastardly crooks where one path is broken into two possible directions, the computer must decide which path is the most likely one to eventually be followed when the game in progress gets there. If the computer guesses correctly, and this can be a big if, it will play the game of an expert competitor. But if it guesses wrong, it may be beaten by the simplest of strategies.

I do not mean to degrade the value of a look-ahead. But I do want to impress upon you that just because a game uses a deep look-ahead to calculate its move does not mean that it will play a good game. Unless the computer looks in the right direction, its search, no matter how deep, could prove to be useless. Even defeating.

The means of reaching a look-ahead depth, then, are potentially more important to the success of the process

than a yardstick measurement of its length. Only by correctly evaluating the strength of a certain game board position, and only by outsmarting its opponent will the computer play the programmer's dream of an unbeatable game. VIP-FLOP is beatable, and I do not say this in despair. Unlike most games on the market, I'll show you how you can try your hand at improving the computer's strategy by writing your own board evaluation routine. For this you will need to understand how the computer performs its look-ahead, and I will devote the rest of this chapter to the process. Even if you do not write your own evaluation, I will show you some simple modifications which may be made to test the computer's playing ability while learning how decisions are made in a game of this type. Certain features have been built in to VIP-FLOP to allow (and I hope encourage) experimenting. Each produces measurable differences in the computer's ability to find the best move.

First, however, please look at the Version Controllers, the routines that control the three possible modes of play, beginning at 035C. VIP-FLOP is subroutine driven to such an extent that the Version Controllers took only 20 minutes to write, debug and test. Rather than write one long routine which jumps around differently depending on

which version was selected, each version was written as a separate program. Each of these programs turned out to be no more than a list of subroutines that control that mode of play. The subs are all the same for each version, but which subs where determine the version to be played. Actually, when a program is subroutined to this extent, each subroutine call may be thought of as one instruction in a specialized language designed to play the game you are programming. You only need to order the subroutines as if you were programming in a new language of your own design to arrive at the final controller. As the Version Controllers are well documented, you should be able to write your own versions with beepers where you want them, and styles of play that suit you, without having to dig into the subroutines themselves or understand how they work. (If you do want to go deeper, those subroutines are equally well documented, too.) Please note also the simplicity of each controller. The computer/computer version is only 20 instructions long occupying 40 bytes of memory! The human/human game controller is even shorter -- 15 instructions for 30 bytes of space run the entire game.

As a suggestion for a new controller, you may consider writing a version controller that keeps a

running time of the human's game and requires you to make all your moves without running out of time, or, you forfeit the game. Maybe your own controller module would make an interesting article for VIPER -- I know I'd be interested in seeing your ideas.

* * *

THE BOARD

The computer keeps its own board in memory and does not see or use the display board at all to figure its moves. The display is strictly output. The computer does not need it to play the game.

The board requires 100 bytes of memory stored at 0800-0863 and is in the form of a matrix 10 bytes long by 10 bytes deep. Of course all memory spaces may be thought of as a continuous stack of bytes while a matrix looks more like a chunk or a box. How is this possible? Let's take a look at the computer's board matrix to understand its construction.

COMPUTER BOARD MATRIX

```
0800  FF FF FF FF FF FF FF FF FF FF
  0A  FF 00 00 00 00 00 00 00 00 FF
  14  FF 00 00 00 00 00 00 00 00 FF
  1E  FF 00 00 00 00 00 00 00 00 FF
0828  FF 00 00 00 01 80 00 00 00 FF
  32  FF 00 00 00 80 01 00 00 00 FF
  3C  FF 00 00 00 00 00 00 00 00 FF
  46  FF 00 00 00 00 00 00 00 00 FF
  50  FF 00 00 00 00 00 00 00 00 FF
  5A  FF FF FF FF FF FF FF FF FF FF
```

Even though the computer board shown here is stored in a continuous manner in memory (as all things), writing it down this way helps to visualize its construction. It looks like an 8 x 8 checkerboard this way, and is easier to think about.

The FF bytes around the board form a border that makes move generation fast and easy by preventing a possible situation where the computer overreaches the edge of the board. The border bytes mark, clearly, the edges of the matrix.

The trick in dealing with a matrix such as this is to find the address of any particular "square" on the board given only the X and Y coordinates to locate that square. In other words, what is the address of the byte that represents the square 3,4? The 3 is the X or horizontal coordinate while the 4 is the Y or vertical. On the sample computer board above,

locate square 3:4. This is at the address 0820.

But how do we get 0820 from the numbers 3:4? The process is quite simple. If you multiply the (Y-1) coordinate times the number of possible horizontal elements in the matrix, you will arrive at the low order of the address of the row corresponding to Y. This address is added to the base address, or beginning address, of the matrix. Using our example, the Y coordinate of 4 minus 1 is 3. Then, 3 x 10 (the number of possible horizontal elements) is equal to 30 decimal which is in turn equal to 1E hex. Notice that the fourth row of the board matrix begins at location 081E. This address is equal to the base address of the board at 0800 + 1E hex, the number we just arrived at.

Now, simply add the value of the X coordinate minus one to 081E and you have the address of the correct "square" in memory! In the example, 3 is the X coordinate. Then, (3-1) + 081E = 0820 which is the address of the square 3:4 on our board.

Any size matrix may be treated in the same way. In fact, your VIP display may be thought of as a large 64 x 32 (or 64 x 64 as for VIP-FLOP) matrix. The subroutine in the Chip-8 interpreter that determines where display patterns go in memory when you program a DXYN instruction, uses the same technique described

here to find the correct address in your display. Its a more complex treatment, but the concept is identical.

The MLS that sets up a computer board in memory is located at 0932, and when called, will generate a bare 100-byte board beginning at a location specified by the "I" memory pointer. To use the sub in your own games, set "I" with an AMMM instruction, call the MLS with an 0932 instruction and you have your board!

The MLS at 0958 places the beginning pieces for VIP-FLOP in the bare matrix. This routine was kept separate from the other in order to permit you to use the MLS at 0932 to create boards for checkers, chess or any game requiring an 8 x 8 grid for play.

The pieces are: 01=white/80=black and you will see the bytes representing VIP-FLOP's starting position on the sample board matrix printed above.

* * *

Now that the computer has its board, what to do with it? The first thing to consider is how the computer performs moves on its board. The MLS located at 0900-0912 with an entry point at 0901, takes the XY coordinates in VA VB and converts them by the process just described into an address which is placed in RA.0, the low part

of the "I" pointer address. The "I" pointer is preset to the base address of the board before calling the sub, a feature that leaves the routine flexible enough to accept a different memory mapping for your own games.

Now that we have a way to reference any board square address, the next step is to make moves on the computer board. This is controlled in VIP-FLOP by the MLS at 0700-073F which makes any move set in VA VB.

You would be correct in thinking that writing two routines to make moves is overcomplicated. VIP-FLOP solves this problem by a routine at 09A4-09BB which will flip flop the board over (again via the preset "I" pointer) allowing the computer to make white moves all the time, even when it is black that is actually moving. It does this by exclusive "OR"ing all pieces with 81 hex which will turn an 80 into an 01 and an 01 into an 80 too. In fact, a black piece is <u>never</u> moved onto the board. Without this feature, the look-ahead would not be possible except with a much greater complexity.

When performing a look-ahead, we need a move generator to create all the possibilities every step of the way. This sub is at 0752 and uses the legal move test at 0864 to generate all the possible moves. This is really simple to do once you have a legal move test that works (and will, of course, prevent a player from

making an illegal move as well.) Test each XY board position against the legal move criteria. If the move is legal, then it is a possibility. If not, do not store it. It's that simple. VIP-FLOP's move generator generates legal moves into a 150-byte area at 0A64, and stores the XY coordinate of the move plus leaving a third byte empty that will later be used to weight the move. The end of the move list is marked with an FF stop byte. The area for the list is longer than necessary, but there was no scramble for memory bytes so I did not worry about the overkill. More is better than not enough!

Another routine is needed to almost complete the building blocks needed for the look-ahead process. This is the MLS at 096F which is an all-purpose sub used to transfer bytes in and out of any areas of memory. It is used by first setting the "I" pointer to the block of data to be moved, then calling the sub with an 096F -- Do MLS instruction followed by the number of bytes to be transferred (to a maximum of 00FF) and finally the address where the data is to go. See 0616-061C for an example of the use of the transfer sub. It will be used to transfer move lists as well as computer boards to and from areas in memory and is a very flexible routine as it transfers data in any direction, as long as the data does not overlap on a forward direction (towards higher

addresses). The data transferred is not altered in its original location by the subroutine.

We now have all but one of the features needed to perform a look-ahead and these same features may be used for any game requiring the process. The missing feature is the board evaluation, but this will be covered a bit later in detail. All we need to know at this point is that the evaluation returns a number representing the strength of white's position on the game board at 0800-0863 storing that number or weight immediately after the move in the move list responsible for creating that board position. The highest weighted move presumably will be the best move to make at that point.

THE LOOK-AHEAD

Computer programmers define a "move" in any two-player game as when both players have each made a move. When one player goes and the other player, too, that is equal to one "move." When only one player goes, it is said to be a "ply" or a half move. Two "plys" equal a whole "move."

VIP-FLOP begins the look-ahead process at 0600 by first flipping the board over. As the computer will be playing the black pieces, it does this because it will

pretend to play white after the board is flipped. (You should begin to see how it is possible for the computer to recommend moves for white and play games with itself. If the board is flipped before calling the BLKMV sub at 0600, the extra flip it gets at the start of the sub will allow the computer to figure white's next move. It's all the same to the computer. It always does everything as if it were playing white.)

After flipping the board, the entire matrix is transferred to a storage area at 0B00-0B63 labeled PERM STORE. (It may be helpful for you to keep your finger on that page so you may refer to it while we discuss the look-ahead.)

The next step is to generate all the possible moves for the board at 0800 which is restored from the saved board at 0B00-0B63. This list, generated at 0A64-0AFF is then transferred to a storage area at 0B64-0BFF.

We now have the original (but flipped) board stored in memory and a primary move list that contains all the possible moves for that board configuration. This leaves the original board at 0800-0863 free for performing the look-ahead.

First an index (V8) is set to 00 to allow the sequential processing of the primary move list at

0B64-0BFF. A move from that list is made on the board (but not the display -- all this happens at high speed in memory). This is performed by the sequence at 0632-0644. The board is again flipped to allow the computer to prepare the responses to the move just made -- remember, the computer always plays white's moves! In between each ply, the board is always flipped so a black move never has to actually be made.

The ply count, the odd number you entered in at the beginning of the game as the level of play, is retrieved from its storage at 07FF and placed in V9 at the beginning of each look-ahead to determine the depth of the search. As V9 is odd, the look-ahead always ends with the opponent's possibilities, but more on this later.

From 0640-0662, the routine just described is in essence repeated, but now a new storage area is used to hold intermediate boards at 0A00-0A63. After generating all moves possible (remember, one of the primary moves has been made to the board which was then flipped, so the resulting move list now contains all possible responses to that one primary move) each move is made on the board and evaluated by the sub at 0786. The best move is then found in the list by a MLS at 09BC that sets "I" to the move with the highest weight

and the move is placed in V0 V1 (the weight is in V2) at location 0658. Next the ply count is decremented and if it is not equal to zero, the program jumps tp 0640 at which point the move in V0 V1 is transferred to VA VB and made on the board. That resulting board is flipped and stored in the intermediate storage area at 0A00-0A63 and the whole process is repeated again and again until the ply count in V9 finally goes to 00 representing the end of the search. It's not that complicated when you plan the process out, but it was enough to get me stuck in the tree I thought I would climb with the greatest of ease! (Moral: computers operate on programs one step at a time. And that's the way a program should be written!)

Now what happens? We have a weight in V2 which represents the opponents best possible move (that is, your move -- you are the computer's opponent) at the end point of our look-ahead. How does that help the computer?

First the weight in V2 is complimented with the instructions at 0664 to 066A. (The other "stuff" there skips the compliment in the event V9 is not equal to zero and is an odd number in which case the look-ahead reached the end of the game but not ending with an opponent's move.) By storing the complimented

weight with the original primary move that started this whole process, we have a representation of the most probable (hopefully) end board position resulting if the computer makes that move. As this weight was complimented, and as it originally stood for the opponent's best move before complimenting, if the computer makes the move in the primary list with the highest weight it will be limiting the opponent to its maximum potential gain.

The computer repeats the entire process for each move in the primary move list finally returning what it"thinks' is the best move to make and that is quite a loop! (Time wise, by the way, the interrupt refresh routine is responsible for slowing down this program somewhat. If the computer were operating without the display on, things would go faster. Hmmm -- now if I bought another VIP, and one operated the display while the other ....)

This completes the look-ahead, but by no means does this represent the ideal way of figuring the computer's move. Several things may be altered to improve the computer's play, and we will turn to these suggestions now.

SOME MORE VARIATIONS

The current strategy just described causes the computer to make a move it decides will limit you to the smallest possible gain. In other words, the computer plays in hopes of maximizing your disadvantage. In English, the computer tries to get tough with you.

This does not represent the best way of figuring a move for VIP-FLOP. I included it with your game as a good starting point. This strategy is a very good way to come up with moves in other games, and many of the chess programs sold today use the process as a basic strategy. Checkers works well this way, too. I did not feel you would be inclined to make changes to your program that worsened the computer's play. So now that you understand this strategy, let's write a better one!

The most obvious thing to do is to cause the look-ahead to end with an examination of the <u>computer's</u> possibilities instead of the opponent's. The result is that the computer will make a move that maximizes its <u>own</u> potential gain rather than minimizing yours. To accomplish this is easy. If the ply count is an <u>even</u> number, the look-ahead will always end with a look at the computer's move. (Or yours if the computer is

figuring a move for white -- remember the board flipping.) The following changes will accomplish this:

```
030E   61FE;V1=FE -- V1=value to "AND" with keypress
0312   8012;V0&V1 -- "AND" variables together (assures
                     V0 is an even number)
```

And in the BLKMV sub:

```
0668   3F00;SK=00 -- If even ply # (or=00) skip next
```

This last change will cause the V2 weight not to be complimented provided the look-ahead ends with a look at the computer's possibilities somewhere down the line. The computer will return the move it feels will eventually give it the most number of pieces.

Now, an extra ½ move will always be made by the look-ahead. If you select level 2 by pressing Key 2 at the game start up, the computer will look 1½ moves ahead. Key 4= a 2½ move look-ahead, Key 6= a 3½ move look-ahead. If you press an odd-numbered key now, the computer automatically changes this to the next lower even number, so you still have the nine levels of possible play. Key 0 selects the identical basic level game as before. At this, the lowest level, the look-ahead is skipped and the computer only makes the move that

nets it the most number of pieces plus a little consideration for the corners which are the best spots to occupy on the board. Knowing this you should be able to trap the computer easily after playing for a while, but the first time I played Level 0 after programming the game, it beat me. How embarrassing!

Regardless of this change, the computer's strategy is at the mercy of the evaluation routine which has the awesome job of weighting board positions. Not only must the routine correctly evaluate the computer's position, but it must assume responsibility for guessing which move you will make next. If you do not make the move the computer expected of you, its entire look-ahead could have been for naught. (I told you that luck had more than a little effect!) Even so, the computer will still be making good moves using a thinking process not wholly unlike the way a human figures a move. You too may not guess your opponent's next move correctly!

One thing is certain. If you can write a better evaluation routine, the computer will play a better game. Here are some ideas.

BOARD EVALUATION

VIP-FLOP's board evaluation is a machine language subroutine located at 08B1-08EA. It is not very complex and contains a built in strategy adjustment which you may make for yet another way to experiment with the effectiveness of the look-ahead.

The very beginning of the sub needs some explanation, though. Please have a look at the MLS Sub Handler at 0918. This sub will cause any machine language subroutine that calls it to begin running with R4 as the program counter instead of R3. This allows other machine language subs that return via a SEP R4 (D4) instruction to be called from both Chip-8 routines and from machine language routines.

The Sub Handler is called using any available register set to 0918 (I have used RC) and calling with a SEP RC (DC) instruction as demonstrated in lines 08B1-08B7. A second DC instruction, without changing the values in RC will now return control to the Chip-8 Interpreter via the second half of the Sub Handler. R3 is free during a machine language sub running in R4 for calls to other subs.

The evaluation routine works by calling the MLS (via R3 and returning via R4) that counts the number of

white and black pieces on the computer's board. The number of white pieces is added to a middle value of 80 (at 0BCB) and the weight is also adjusted for corner positions with the subroutine calls from 08D0 to 08E4. The Adjust sub at 08EB adds a value of 5 to the weight for each corner occupied by white. (I know I've said it before, but this is because all moves, evaluations, etc., are always done for white -- even if black is the piece moving.) You may want to experiment with different weights for the corners by adjusting the value of the byte at 08F1 up or down to see what happens.

As currently programmed, this is the computer's total evaluation strategy. It counts the number of white pieces, adds an adjustment for the corners and returns the resulting weight as Chip-8's V2 value with the instructions at 08E5-08E9 before returning, via the Sub Handler, with the DC SEP RC instruction at 08EA.

A built-in strategy change may be made at locations 08CD-08CF. Notice that two bytes are printed at each of these locations in the program listing. The E2 bytes, which are supplied with the tape of the game, function as NOPS (No operation instructions). If you enter the others, 8F F5 52 however, the evaluation will now subtract the number of black pieces <u>from</u> the number of whites adding this (in the process) to the middle weight

of 80 to arrive at a value representing the difference in pieces between white and black. I confess to not having fully experimented with this change, but there it is for you to try. I'd love to know what you find out. (VIPER would, too, I bet!)

The only requirement you must stick to if you want to attempt a totally new approach to an evaluation sub is to return a weight as the value of Chip-8's V2. (This is at 02F2 -- see appendix for explanations of Chip-8 Interpreter modifications that were made.) The sub must be in machine language and begin at 08B1, but you are otherwise free to experiment. If you have a friend with a VIP, you could even play your strategies against each other as a contest to see who can write the best routine! Cosmac computer clubs, how about a programming tournament?

One idea would be to change strategies during, say, the middle of the game or to write a special killer routine that plays a brutal end game. Another would be to adjust strategies accoding to what the opponent does and try to give the computer a firmer base on which to decide your next most probable move.

I have wondered if the evaluation could use a look-ahead of its own as part of the evaluation. Can you figure a way out of the resulting Catch-22 situation?

(There I am in that tree again.)

Maybe pattern recognition could play a role in the process. The computer could be taught to build up blocks from corners or work on traps that assure it of an eventual gain.

How about a program that learns to play a better game the more it plays? That would truly be a challenge -- with a pot-of-gold satisfaction value if you get it to work.

I could continue. My bane is I am never satisfied and to be perfectly honest, I hope you never are either. Question what the "experts" tell you (yes, me too, though I do not want to say I am an expert, I hope you'll tear my things apart and put 'em back together the way you want 'em. Nothing pleases me more than to see an improvement I hadn't even thought of. Damned clever these programmers!). Experiment, write and rewrite, test and test again, enjoy your computer.

It's an education you will not ever regret.

VARIABLE ASSIGNMENT

V0 - Various--V0 passes values to and from most subs
V1 - " " "
V2 - " " "
V3 - " " "
V4 - Turn indicator/01=white; 80=black
V5 -
V6 -
V7 - Version number (01=H-C; 02=C-C; 03=H-H)
V8 - Index to primary move list for computer's move/other uses
V9 - Level of play (ply count down in computer's move/other uses
VA - VX board coordinate 1-8 (not display)
VB - VY board coordinate 1-8 (not display)
VC - VX display coordinate
VD - VY display coordinate
VE - Index to generated move lists
VF - Various, flags, etc.

MEMORY MAP

0000-02FF -- Chip-8 Interpreter - two-page display
0300-0BFF -- Program, subs, data
0C00-0DFF -- Character set (modified)
0E00-0FFF -- Two pages for display refresh

ROUTINES AND DATA

0300-035A -- Initializing, display set-up, game prompting
035C-039E -- Version #1 Controller (H-C games) (VERS1)
03A4-03CA -- Version #2 Controller (C-C games) (VERS2)
03CC-03E8 -- Version #3 Controller (H-H games) (VERS3)

0400-0410 -- Index VC VD per VA VB sub (INDEX)
0412-0426 -- Draw playing board sub (DRAW)
0428-0434 -- Display piece sub (PIECE)
0436-045A -- Set-up VIP-FLOP sub (OTHLO)
045C-04A0 -- Make move sub (MKMOV)
04A2-04C8 -- Change line (part of MKMOV sub) (CHNGE)
04CA-04D2 -- Timer sub (TIMER)
04D4-04E4 -- Beeper sub (BEEPR)

```
0500-050C -- Win detector sub                                  (WIN)
050E-0540 -- End game/restart routine                          (ENDGM)
0542-0550 -- Get keypress sub                                  (GETKY)
0552-0566 -- Blink move sub                                    (BLINK)
0568-057A -- Shimmer move sub                                  (SHIMR)

0600-0688 -- Black moves controller sub                        (BLKMV)
068A-0696 -- Message center sub                                (MCENT)
0698-069C -- Print sub                                         (PRINT)
069E-06C2 -- Numbers conversion sub                            (NUMBS)
06C4-06D6 -- Turn switcher sub                                 (TURNS)
06D8-06EE -- Display move in VA VB sub                         (DSPMV)
06F4-06FE -- Bit patterns for display    (WHITE)(BLACK)(LINE)

0700-073F -- MLS--Make move
0740-0750 -- Change line (part of MLS Make Move)
0752-0784 -- Move Generator sub                                (GENER)
0786-07AE -- Evaluation controller sub                         (EVCNT)
07B0-07DA -- Input move sub                                    (INPUT)
07DC-07FA -- Scoring sub                                       (SCORE)
07FF      -- Storage byte - level of play

0800-0863 -- 100 bytes for computer board                      (BOARD)
0864-08B0 -- MLS--Test legal move in VA VB
08B1-08EA -- MLS--Board evaluation
08EB-08F5 -- MLS--Adjust (for  corners)

0900-0912 -- MLS--Reference RA to board address
0918-0923 -- MLS--Sub handler
0924-0931 -- MLS--Search sub
0932-0957 -- MLS--Computer 8x8 grid set-up
0958-096E -- MLS--Beginning positions set-up
096F-097D -- MLS--Transfer
097E-09A3 -- MLS--Count whites/Blacks
09A4-09BB -- MLS--Flip flop board
09BC-09E0 -- MLS--Get best move

0A00-0A63 -- 100 bytes - Temp Store
0A64-0AFF -- 156 bytes - Move list #1

0B00-0B63 -- 100 bytes - Perm Store
0B64-0BFF -- 156 bytes - Move list #2

0C00-0DFF -- Character set (modified as follows)
0D6C-0DFF -- ASCII Message strings
```

PROGRAM LISTING

```
0300  BEGIN :AD6C ;MSGS  -- Set "I" to ASCII prompt messages
  02          268A  MCENT -- Do sub -- display 4 text lines
  04          60E0 ;V0=E0 -- V0 passes value to timer sub
  06          24CA  TIMER -- Do sub -- wait before continuing
  08          0230 ;ERASE -- Erase screen (not 00E0 for 2-page
                                                  interrupt)
  0A          268A ;MCENT -- Do sub -- display 4 text lines
  0C          F00A ;V0=KY -- V0=next key press - level of play
  0E          6101 ;V1=01 -- V1=value to "OR" with key press

0310          3000 ;SK=00 -- If Key 0 pressed, skip next
  12          8011 ;V0/V1 -- Logic "OR" 01/key pressed (assures
                                                      odd number)
  14          A7FF ;PLY   -- Set "I" to ply count storage
  16          F055 ;PUT   -- Store key press as level of play
                                          (look ahead depth)
  18          ADBE ;MSGS  -- Set "I" to next ASCII prompt string
  1A          0230 ;ERASE -- Clear screen
  1C          268A ;MCENT -- Do sub -- display 4 text lines
  1E  VERSN :F70A ;V7=KY -- V7=next key press - version?

0320          4700 ;SK≠00 -- Skip next if key press in V7 ≠ 00
  22          131E  VERSN -- Jump to get a valid entry
  24          6F03 ;VF=03 -- VF holds limiting value of 03
  26          8F75 ;VF-07 -- Subtract 03 - key press
  28          3F01 ;SK= + -- If positive or zero, skip (0< keypress ≤ 3)
  2A          131E  VERSN -- Jump to get a valid entry
  2C          0230 ;ERASE -- Clear screen, prepare for game
  2E          A800 ;BOARD -- Set "I" to area for computer board

0330          0932 ;MLS   -- Do MLS -- create 8 x8 grid matrix @ I
  32          A800 ;BOARD -- Reset "I" to computer grid
  34          0958 ;MLS   -- Do MLS -- set-up for VIP-FLOP
  36          2412  DRAW  -- Do sub -- display 8 x 8 grid (empty)
  38          2436  OTHLO -- Do sub -- display set-up for VIP-FLOP
  3A          6401 ;V4=01 -- Initialize turn indicator V4=01=White
  3C          6C00 ;VC=00 -- VC is VX for display white/black moves
  3E          6D00 ;VD=00 -- VD  " VY          "          "          "

0340          ADF4 ;MSGS  -- Set "I" to ASCII message string
  42          2698 ;PRINT -- Do sub -- print "WHITE"
  44          7D08 ;VD+08 -- Add 08 to VY coordinate for next line
  46          2698  PRINT -- Do sub -- print "MOVES"
  48          6D20 ;VD=20 -- Set VD=VY for scoring set-up
  4A          A53C ;WHITE -- Set "I" to ASCII for white piece =
  4C          2698 ;PRINT -- Do sub -- print the symbol & = sign
  4E          7D07 ;VD+07 -- Add 07 to VY for next line
```

```
0350          2698 ;PRINT -- Do sub -- print the symbol & = sign
  52          27E4 ;SCOR2 -- Do sub -- initialize the score
  54          4703 ;SK≠03 -- Skip if version #3 not selected
  56          13CE  VERS3 -- Jump to human/human game
  58          4702 ;SK≠02 -- Skip if version #2 not selected
  5A          13A4  VERS2 -- Jump to computer/computer game
```

VERSION #1 CONTROLLER (H-C)

```
035C  VERS1 ;27B0 ;INPUT -- Do sub -- input move
  5E          3000 ;SK=00 -- If V0=00, then move was selected

0360          138C  H-C3  -- Jump to allow computer to go first
  62          137A  H-C2  -- Jump to begin game
  64  H-C1    ;26DA  DSPMV -- Do sub -- erase move displayed
  66          27B0 ;INPUT -- Do sub -- input move
  68          300C ;SK=0C -- If Key C pressed, skip to next part
  6A          137A  H-C2  -- Jump to continue past next part
  6C          A800 ;BOARD -- Set "I" to computer board
  6E          09A4 ;FLIP  -- Do MLS -- flip flop board

0370          2600  BLKMV -- Do sub -- suggest move for white
  72          A800 ;BOARD -- Set "I" to computer board
  74          09A4  FLIP  -- Do MLS -- flip flop board back
  76          24D4  BEEPR -- Do sub -- beeper signals ready
  78          26DA  DSPMV -- Do sub -- display suggested move
  7A  H-C2    ;2568  SHIMR -- Do sub -- display move selected
  7C          300E ;SK=0E -- If Key E was pressed, skip next
  7E          1364  H-C1  -- Jump to cancel move--Key F pressed

0380          245C  MKMOV -- Do sub -- make the move
  82          26DA  DSPMV -- Do sub -- erase move (X;X) displayed
  84          27DE  SCORE -- Do sub -- update the score
  86          2500  WIN   -- Do sub -- check for win
  88          3F00 ;SK=00 -- If VF=00, skip to continue game
  8A          150E ;ENDGM -- Else jump to end game/restart
  8C  H-C3    ;26C4  TURNS -- Do sub -- display "BLACK MOVES"
  8E          2600  BLKMV -- Do sub -- figure black's move

0390          24D4  BEEPR -- Do sub -- sound beeper
  92          26D8  DSPMV -- Do sub -- display black's move
  94          2552  BLINK -- Do sub -- blink move
  96          245C  MKMOV -- Do sub -- make the move
  98          27DE  SCORE -- Do sub -- update the score
  9A          2500  WIN   -- Do sub -- check for win
  9C          3F00  SK=00 -- If VF=00, skip to continue
  9E          150E ;ENDGM -- Else jump to end game/restart
```

```
03A0          26C4  TURNS -- Do sub -- display "WHITE MOVES"
  A2          1364  H-C1  -- Jump to get white's move
```

VERSION #2 (C-C)

```
03A4  VERS2 :3401 ;SK=01 -- If V4 turn indicator =01, skip next
  A6          13AC  C-C1 -- Jump past flip flop
  A8          A800 ;BOARD -- Set "I" to computer board
  AA          09A4 ;FLIP  -- Do MLS -- flip flop board
  AC  C-C1  :2600 ;BLKMV -- Do sub -- figure next move
  AE          3401 ;SK=01 -- If V4=01 (white), skip next

03B0          13B6  C-C2 -- Jump past flip flop
  B2          A800 ;BOARD -- Set "I" to computer board
  B4          09A4 ;FLIP  -- Do MLS -- flip flop board back
  B6  C-C2  :26D8 ;DSPMV -- Do sub -- display move (X:X)
  B8          24D4 ;BEEPR -- Do sub -- sound beeper
  BA          2552 ;BLINK -- Do sub -- blink the piece (may be replaced)
  BC          245C ;MKMOV -- Do sub -- make the move
  BE          26D8 ;DSPMV -- Do sub -- display move (X:X) to erase

03C0          27DE ;SCORE -- Do sub -- update the score
  C2          2500 ;WIN   -- Do sub -- check for win
  C4          3F00 ;SK=00 -- If VF=00, skip to continue
  C6          150E ;ENDGM -- Else jump to end game/restart
  C8          26C4 ;TURNS -- Do sub -- display white/black moves
  CA          13A4  VERS2 -- Jump for next move
```

VERSION #3 (H-H)

```
03CC  H-H1  :26DA ;DSPMV -- Do sub -- erase move (X:X) to cancel
  CE  VERS3 :27B0  INPUT -- Do sub -- input player's move

03D0          3000 ;SK=00 -- Skip if move selected
  D2          13CE  VERS3 -- Jump to get valid (≠ 0C!) entry
  D4          2568  SHIMR -- Do sub -- shimmer piece
  D6          300E ;SK=0E -- Skip if Key E was pressed
  D8          13CC  H-H1  -- Jump to cancel move
  DA          245C ;MKMOV -- Do sub -- make the move
  DC          26DA ;DSPMV -- Do sub -- erase move (X:X)
  DE          27DE ;SCORE -- Do sub -- update the score
```

```
03E0          2500 ;WIN    -- Do sub -- check for win
  E2          3F00 ;SK=00  -- If VF=00, skip to continue
  E4          150E ;ENDGM  -- Else jump to end game/restart
  E6          26C4 ;TURNS  -- Do sub -- switch white/black moves
  E8          13CE  VERS3  -- Jump for next player's move

03EA-03FF -- Not used -- available for expansion versions
```

INDEX VC VD PER MOVE VA VB (FOR DISPLAY)

```
0400  INDEX :8CAE ;SHL    -- Multiply VA x 4 by shifting left x 2 and
  02          8CCE ;SHL    -- Put result in VC (VX display)
  04          8CA4 ;VC+VA  -- Add to complete a multiply x 5
  06          7C13 ;VC+13  -- Add 13 hex to adjust X coordinate
  08          8DBE ;SHL    -- Multiply VB x 8 by shifting left x 3 and
  0A          8DDE ;SHL    -- Put result in VD (VY display)
  0C          8DDE ;SHL    --       "              "
  0E          7DFA ;VD-06  -- Subtract 06 to adjust Y coordinate
0410          00EE ;RET    -- Return with VC VD = XY coordinate
```

DRAW PLAYING BOARD

```
0412  DRAW   :A6FC  LINE   -- Set "I" to blank space line
  14          6D02 ;VD=02  -- VD is VY for display
  16  DRAW1  :6C18 ;VC=18  -- VC is VX  "     "
  18  DRAW2  :DCD4 ;SHOW   -- Display line @ VC VD
  1A          7C05 ;VC+05  -- Increment X = next in row
  1C          3C40 ;SK=40  -- When VC = 40, skip next
  1E          1418 ;DRAW2  -- Jump to do one row

0420          7D08 ;VD+08  -- Increment Y = next row
  22          3D42 ;SK=42  -- When VD = 42, skip next
  24          1416  DRAW1  -- Jump to do all rows
  26          00EE ;RET    -- Return
```

DISPLAY PIECE

```
0428  PIECE :A6FC ;LINE   -- Set "I" to bit pattern for blank line
  2A          DCD4 ;SHOW   -- Display to erase line
  2C  PIEC1  :A6F4 ;WHITE  -- Set "I" to bit pattern for white piece
  2E          3401 ;SK=01  -- If V4 turn indicator=01=white, skip next
```

```
0430          A6F8 ;BLACK -- Set "I" to bits for black piece (V4=80)
  32          DCD4 ;SHOW  -- Display piece
  34          00EE ;RET   -- Return
```

SET-UP VIP-FLOP

```
0436  OTHL0 :6401 ;V4=01 -- Turn indicator = 01 = white
  38          6A04 ;VA=04 -- Move XY = 04:04 square
  3A          6B04 ;VB=04 --    "             "
  3C  OTHL1 :2400  INDEX -- Do sub -- set VC VD to board square XY's
  3E          2428  PIECE -- Do sub -- display piece @ VC VD

0440          7A01 ;VA+01 -- Add to X      For
  42          7B01 ;VB+01 --  "   " Y        square 05:05
  44          3A06 ;SK=06 -- If VA=06, skip
  46          143C  OTHL1 -- Loop to do next piece
  48          6480 ;V4=80 -- Turn indicator = 80 = black
  4A          6A04 ;VA=04 -- Move XY = 04:05 square
  4C          6B05 ;VB=05 --    "             "
  4E  OTHL2 :2400  INDEX -- Do sub -- set VC VD to board square XY's

0450          2428  PIECE -- Do sub -- display piece @ VC VD
  52          7A01 ;VA+01 -- Add to X                For
  54          7BFF ;VB-01 -- Subtract from Y           square 05:04
  56          3A06 ;SK=06 -- Skip if VA = 06
  58          144E  OTHL2 -- Loop to do next piece
  5A          00EE ;RET   -- Return
```

MAKE MOVE SUB (BLACK OR WHITE) DISPLAY

```
045C  MKMOV :3480 ;SK=80 -- If turn indicator=80=black, skip into
                                                        next part
  5E          1464  MKM1  -- Jump to skip board flip on white's turn

0460          A800 ;BOARD -- Set "I" to computer board
  62          09A4  FLIP  -- Do MLS -- flip flop board on black's turn
  64  MKM1  :2400  INDEX -- Do sub -- set VC VD to XY board square
  66          2428  PIECE -- Do sub -- display piece @ VC VD
  68          6001 ;V0=01 -- V0 = 01 for beep
  6A          F018 ;TONE  -- Sound tone (piece moved onto square)
  6C          88A0 ;V8=VA -- Save VA VB move in V8 V9
  6E          89B0 ;V9=VB --    "        "         "
```

```
0470          62FF ;V2=FF -- Cycle index #2 (VY)
  72  MKM2   :61FF ;V1=FF --   "     "   #1 (VX)
  74  MKM3   :8A14 ;VA+V1 -- Add to VX for next in line
  76          8B24 ;VB+V2 --   "    VY      "       "
  78          A800  BOARD -- Set "I" to computer board
  7A          0901  REFER -- Do MLS -- Set "I" to board square
  7C          F065 ;GET   -- V0 = piece @ M(I)
  7E          4001  SK≠01 -- Skip ≠ 01 white (end of that row)

0480          14A2  CHNGE -- Go change row
  82          4080  SK≠80 -- Skip ≠ 80 black (not end of row yet)
  84          1474  MKM3  -- Jump to continue this row check
  86  MKM4   :8A80 ;VA=V8 -- Restore VA VB values held in V8 V9
  88          8B90 ;VB=V9 --   "          "          "
  8A          7101 ;V1+01 -- Add to VX cycle index--next direction
  8C          3102 ;SK=02 -- But skip when V1=02
  8E          1474  MKM3  -- Jump to loop on another row

0490          7201 ;V2+01 -- Add to VY cycle index--next direction
  92          3202 ;SK=02 -- But skip when v2=02
  94          1472  MKM2  -- Jump to loop on 3 more rows
  96          0700  MKMOV -- Do MLS -- make move on computer board
  98          3480 ;SK=80 -- If V4 turn indicator=80=black, skip next
  9A          00EE ;RET   -- Return (white)
  9C          A800 ;BOARD -- Set "I" to computer board
  9E          09A4  FLIP  -- Do MLS -- flip flop board back to original
04A0          00EE ;RET   -- Return (black)                      state
```

(CHANGE LINE)

```
04A2  CHNGE :8A80 ;VA=V8 -- Reset VA VB from values held in V8 V9
  A4          8B90 ;VB=V9 --   "          "          "
  A6  CHNG1  :6020 ;V0=20 -- V0 passes value to timer sub
  A8          24CA  TIMER -- Do sub -- wait between changes
  AA          8A14 ;VA+V1 -- Add cycle indexes to move along row
  AC          8B24 ;VB+V2 --   "          "          "
  AE          A800 ;BOARD -- Set "I" to computer board

04B0          0901  REFER -- Set "I" to board square at VA VB
  B2          F065 ;GET   -- V0 = piece @ M(I)
  B4          4001 ;SK≠01 -- If not white, skip to change piece
  B6          1486 ;MKM4  -- Jump back, change complete
  B8          2400  INDEX -- Do sub -- index VC VD to board square
  BA          6081 ;V0=81 -- 81 will compliment 01/80 via XOR
  BC          8403 ;XOR   -- Compliment white/black indicator
  BE          242C  PIEC1 -- Do sub -- display piece to erase opponent
```

```
04C0          8403 ;XOR    -- Compliment V4 back again
  C2          242C  PIEC1  -- Do sub -- display changed color
  C4          6001 ;V0=01  -- V0 is value for tone generator
  C6          F018 ;TONE   -- Sound tone - beep
  C8          14A6  CHNG1  -- Jump to change next or exit
```

TIMER SUB

```
04CA  TIMER :F015 ;TI=V0   -- Set timer to value in V0
  CC  TIME  :F007 ;V0=TI   -- Set V0 = current timer value
  CE          3000 ;SK=00  -- When V0=00, skip to exit

04D0          14CC  TIME   -- Jump to loop till timer = 00
  D2          00EE ;RET    -- Return
```

BEEPER SUB

```
04D4  BEEPR :6109 ;V1=09   -- V1 = loop count (# beeps) of 09
  D6  BEEP  :6001 ;V0=01   -- V0 = value for tone
  D8          F018 ;TONE   -- Sound tone--beep
  DA          6006 ;V0=06  -- V0 passes value to timer sub
  DC          24CA  TIMER  -- Do sub -- wait between beeps
  DE          71FF ;V1-01  -- Add FF to loop count to subtract 01

04E0          3100 ;SK=00  -- When V1=00, skip to exit
  E2          14D6 ;BEEP   -- Jump to beep again
  E4          00EE ;RET    -- Return
```

04E6-04FF -- Not used

WIN DETECTOR

```
0500  WIN   :A800 ;BOARD   -- Set "I" to computer board
  02          097E ;COUNT  -- Do MLS -- count # pieces
  04          8014 ;V0+V1  -- Add whites + blacks (in V0 V1)
  06          6F00 ;VF=00  -- Set VF=00 = continue flag
  08          4040 ;SK≠40  -- If ≠ 40 pieces (64 decimal), skip to exit
  0A          6F01 ;VF=01  -- Else set VF=01 to flag end of game
  0C          00EE ;RET    -- Return
```

WIN BLOCK/RESTART (NOT A SUBROUTINE)

```
050E  ENDGM :A800 ;BOARD -- Set "I" to computer board

0510        097E ;COUNT -- Do MLS -- count # pieces
  12        9010 ;SK≠   -- If V0≠V1, skip next
  14        152A  END   -- Jump past win block for tie games
  16        6C08 ;VC=00 -- VC is VX for win block display
  18        6D1F ;V0=1F -- VD  " VY  "      "       " (white)
  1A        8015 ;V0-V1 -- Subtract whites-blacks to find winner
  1C        3F01 ;SK= + -- If positive, white > black & is winner
  1E        6D26 ;VD=26 -- Else reset VD (VY) for black block

0520        A534  BLOCK -- Set "I" to bit pattern for win block
  22  FLASH :DCD7 ;SHOW  -- Display 1/9 block
  24        7C01 ;VC+01 -- Add 01 to X coordinate
  26        3C11 ;SK=11 -- Skip next when VC = 11
  28        1522 ;FLASH -- Jump to display whole block
  2A  END   :6000 ;V0=00 -- V0=00 for keypress test
  2C        E09E ;SK=KY -- If Key 0 pressed, (=V0), skip next
  2E        150E  ENDGM -- Jump to flash block (or not on ties)

0530        0230 ;ERASE -- Erase display to prepare for restart
  32        1300  BEGIN -- Jump to beginning for new game
  34  BLOCK :8080 ;BITS  -- Bit pattern for win block
  36        8080        --   "          "       "
  38        8080        --   "          "       "
  3A        8000        --   "          "       "
  3C  WHITE :1F3D ;ASCII -- String for scoring set-up
  3E  BLACK :001E        --   "          "       "
0540        3D00        --   "          "       "
```

GET KEYPRESS

```
0542  GETKY :F00A ;V0=KY -- Let V0=value next keypress (waits)
  44        400C  SK≠0C -- If not Key C, skip next
  46        00EE ;RET   -- Key C pressed
  48        6F08 ;VF=08 -- VF is limiting value (0-8)
  4A        8F05 ;VF-V0 -- Subtract 08 - keypress
  4C        3F01 ;SK= + -- If positive, skip next (V0≤ 8)
  4E        1542  GETKY -- Else jump to get a valid entry
0550        00EE ;RET   -- Return (V0= 0-8)
```

BLINK MOVE (FOR COMPUTER)

```
0552  BLINK :2400 ;INDEX -- Do sub -- set VC VD per move in VA VB
  54          6E08 ;VE=08 -- VE is loop count of 08
  56  BLIN1 :2428 ;PIECE -- Do sub -- display a piece at VC VD
  58          6001 ;V0=01 -- V0 is value for tone duration
  5A          F018 ;TONE  -- Sound tone (beep each blink)
  5C          6010 ;V0=10 -- V0 passes value to timer sub
  5E          24CA ;TIMER -- Do sub -- wait before continuing

0560          7EFF ;VE-01 -- Subtract 01 from loop count
  62          3E00 ;SK=00 -- When VE=00, skip to exit
  64          1556  BLIN1 -- Jump to blink piece again
  66          00EE ;RET   -- Return-- piece is off
```

SHIMMER MOVE (HUMAN OR COMPUTER)

```
0568  SHIMR :2400 ;INDEX -- Do sub -- set VC VD per move in VA VB
  6A  SHIM1 :2428 ;PIECE -- Do sub -- display a piece at VC VD
  6C          2428 ;PIECE -- Do sub -- (assures that piece is off on exit)
  6E          600E ;V0=0E

0570          E0A1 ;≠SKIP -- If key pressed ≠ 0E, skip
  72          00EE ;RET   -- Return--V0=0E signals enter move
  74          600F ;V0=0F
  76          E09E ;=SKIP -- If key pressed = 0F, skip to exit
  78          156A  SHIM1 -- Jump to reshimmer piece
  7A          00EE ;RET   -- Return--V0=0F signals cancel move
```

057C-05FF-- Not used

BLACK MOVES

```
0600  BLKMV :A800 ;BOARD -- Set "I" to computer board
  02          09A4 ;FLIP  -- Do MLS -- flip flop board
  04          096F ;TRANS -- Do MLS -- transfer board to save
  06          0064 ; #    -- # bytes for transfer (64=100 decimal)
  08          0B00 ;PERM  -- Address where board is saved
  0A          A800 ;BOARD -- Set "I" to computer board
  0C          096F ;TRANS -- Do MLS -- transfer
  0E          0064 ; #    -- # bytes
```

```
0610          0A00 ;TEMP  -- Put board in temporary storage too
  12          2752 ;GENER -- Do sub -- generate move list
  14          2786 ;EVCNT -- Do sub -- evaluate move list
  16          AA64 ;LIST  -- Set "I" to move list
  18          096F ;TRANS -- Do MLS -- transfer primary move list
  1A          0096 ; #    -- # bytes (96=150 decimal)
  1C          0B64 ;LIST2 -- Address where move list (prime)stored
  1E          6800 ;V8=00 -- Set index to primary moves = 00

0620  BLK1   :A7FF ;V9    -- Set "I" to ply count @ 07FF
  22          F065 ;GET   -- Get ply count
  24          4000 ;SK≠00 -- Skip next if count ≠ 00 (first level)
  26          167A ;BLK4  -- Jump to exit
  28          8900 ;V9=V0 -- Let V9=ply count (for look ahead depth)
  2A          AB00 ;PERM  -- Set "I" to permanent board in memory
  2C          096F ;TRANS -- Do MLS -- transfer
  2E          0064 ; #    --         -- 100 bytes

0630          0800 ;BOARD --         -- back to 0800 (restore original
  32          AB64 ;LIST2 -- Set "I" to primary list              board)
  34          F81E ;I+V8  -- Add index to next move
  36          F165 ;GET   -- V0 V1 = move
  38          40FF ;SK≠FF -- If not FF stop byte, skip to continue
  3A          167A  BLK4  -- Else jump to exit
  3C          6F02 ;VF=02
  3E          FF18 ;TONE  -- Beep between searches

0640  BLK2   :8A00 ;VA=V0 -- Let VA VB = XY move
  42          8B10 ;VB=V1 --     "          "
  44          0700 ;MKMOV -- Do MLS -- make move
  46          A800 ;BOARD -- Set "I" to board
  48          09A4 ;FLIP  -- Do MLS -- flip flop board for response move
  4A          096F ;TRANS -- Do MLS -- transfer
  4C          0064 ; #    --         -- 100 bytes
  4E          0A00 ;TEMP  -- To temporary storage

0650          2752 ;GENER -- Do sub -- generate move list
  52          2786 ;EVCNT -- Do sub -- evaluate move list
  54          AA64 ;LIST  -- Set "I" to move list
  56          09BC ;GTBST -- Do MLS -- Set "I" to best move in list
  58          F265 ;GET   -- V0 V1=move; V2=weight
  5A          40FF ;SK≠FF -- If = FF, then look ahead hits end game
  5C          1664 ;BLK3  -- Early exit on hitting end game
  5E          79FF ;V9-01 -- Subtract 01 from ply count

0660          3900 ;SK=00 -- Skip ply count = 00
  62          1640 ;BLK2  -- Jump to do next ply
  64  BLK3   :60FF ;V0=FF -- For complimenting weight
  66          8F96 ;SHR   -- Shift V9 right to test if even or odd
```

```
0668          3F01 ;SK=01  -- If odd ply #, skip next
  6A          8203 ;XOR    -- Exclusive OR weight to compliment
                                 (even # plys)
  6C          AB64 ;LIST2  -- Set "I" to saved move list (primary)
  6E          F81E ;I+V8   -- Index to current start move

0670          F165 ;GET    -- This advances "I" to move's weight
  72          8020 ;V0=V2  -- V0 holds weight for storing
  74          F055 ;PUT    -- Store weight with move
  76          7803 ;V8+03  -- Add 3 to index
  78          1620  BLK1   -- Jump for next move
  7A  BLK4   :A800 ;BOARD  -- Set "I" to computer board
  7C          09A4 ;FLIP   -- Do MLS -- flip flop to original state
  7E          AB64  LIST2  -- Set "I" to saved weight list (Perm)

0680          09BC ;GTBST  -- Do MLS -- Set "I" to best move in list
  82          F165 ;GET    -- V0 V1 = move
  84          8A00 ;VA=V0  -- Let VA = X coordinate move
  86          8B10 ;VB=V1  --  "  VB = Y     "          "
  88          00EE ;RET    -- Return (move in VA VB)
```

MESSAGE CENTER ("I" SET BY CALLER)

```
068A  MCENT :6C00 ;VC=00  -- VC is VX for display
  8C          6D00 ;VD=00  -- VD is VY for display
  8E  MCEN1 :2698 ;PRINT   -- Do sub -- print the message

0690          7D10 ;VD+10  -- Add 01 to VY for next line
  92          3D40 ;SK=40  -- But skip when VD goes to 40 hex
  94          168E  MCEN1  -- Loop to show next line
  96          00EE ;RET    -- Return
```

PRINT SUB

```
0698  PRINT :0244 ;MLS     -- Do MLS -- call messager
  9A          DCD6 ;SHOW   -- DXYN for use by messager
  9C          00EE ;RET    -- Return (VC VD not changed)
```

NUMBERS CONVERSION

```
069E  WORK   :0000 ;4BYT  -- Work area for conversion

06A0          0000
  A2  NUMBS  :A69E ;WORK  -- Set "I" to work area above
  A4          F033 ;3-DD  -- Let V0=3 digit decimal number @ I
  A6          F265 ;GET   -- Let V0:V2 = those digits
  A8          6330 ;V3=30 -- V3 holds constant of 30
  AA          8031 ;V0/V3 -- Logical "OR" number/30 hex
  AC          8131 ;V1/V3 -- to form ASCII equivalent
  AE          8231 ;V2/V3 -- of the 3 digits

06B0          6300 ;V3=00 -- Let V3=00 null character to end string
  B2  NUMB1  :3030 ;SK=30 -- If V0=30 (00 in ASCII) skip into next
  B4          16BE ;NUMB2 -- Jump past next part
  B6          8010 ;V0=V1 -- Move number up to delete leading zeroes
  B8          8120 ;V1=V2 --   "                "                "
  BA          8230 ;V2=V3 --   "                "                "
  BC          16B2  NUMB1 -- Jump to recheck V0
  BE  NUMB2  :A2F0 ;V0    -- Set "I" to V0 in memory

06C0          2698  PRINT -- Do sub -- print number
  C2          00EE ;RET   -- Return
```

TURN SWITCHER

```
06C4  TURNS  :ADE8 ;MSGS  -- Set "I" to ASCII message "WHITE/BLACK"
  C6          4480 ;SK≠80 -- Skip next if V4 turn indicator ≠ 80 (black)
  C8          ADEE ;MSGS  -- Set "I" to ASCII message "BLACK/WHITE"
  CA          6C00 ;VC=00 -- VC is VX for turn message display
  CC          6D00 ;VD=00 -- VD is VY  "         "          "
  CE          2698 ;PRINT -- Do sub -- print the message (to erase)

06D0          2698 ;PRINT -- Do sub -- print the message (to show new)
  D2          6081 ;V0=81 -- V0 holds value for complimenting V4
  D4          8403 ;XOR   -- Exclusive "OR" V4:81 to switch 01↔80
  D6          00EE ;RET   -- Return (display and V4 "RIGHT")
```

DISPLAY MOVE IN VA VB

```
06D8  COLON :3A00 ;ASCII -- ASCII for (:) and a 00 stop byte
  DA  DSPMV :6C04 ;VC=04 -- VC is VX for move display
  DC         6D10 ;VD=10 -- VD  " VY  "          "
  DE         80A0 ;V0=VA -- Let V0 = move X coordinate in VA

06E0         26A2 ;NUMB3 -- Do sub -- display number
  E2         A6D8  COLON -- Set "I" to ASCII for colon
  E4         7C04 ;VC+04 -- Add 4 to VX to print colon
  E6         2698 ;PRINT -- Do sub -- print the colon
  E8         7C04 ;VC+04 -- Add 4 to VX for second number
  EA         80B0 ;V0=VB -- Let V0 = move Y coordinate in VB
  EC         26A2 ;NUMBS -- Do sub -- display number
  EE         00EE ;RET   -- Return

06F0-06F3 -- Not used
```

BIT PATTERNS & DATA

```
06F4  WHITE :60F0 ;BITS  -- Bit pattern white piece
  F6         F060        --      "         "
  F8  BLACK :6090 ;BITS  -- Bit pattern black piece
  FA         9060        --      "         "
  FC  LINE  :0060 ;BITS  -- Bit pattern blank space (a line)
  FE         0000        --      "         "
```

MLS - MAKE MOVE (COMPUTER BOARD ONLY)

```
0700  F8  LDI       ;Set RC = address sub handler
  01  09            ;    to switch PC to R4
  02  BC  PHI   RC  ;   "      "      "
  03  F8  LDI       ;   "      "      "
  04  18            ;   "      "      "
  05  AC  PLO   RC  ;   "      "      "
  06  DC  SEP   RC  ;Do MLS -- this sub runs in R4
  07  F8  LDI       ;Set RA.1 = base address computer board
  08  08            ;   "      "        "                " @ 0A00
  09  BA  PHI   RA  ;   "      "        "                  "
```

```
070A  F8  LDI      ;Set R3 = address refer sub @ 0901
  0B  09           ;    "              "              "
  0C  B3  PHI  R3  ;    "              "              "
  0D  F8  LDI      ;    "              "              "
  0E  01           ;    "              "              "
  0F  A3  PLO  R3  ;    "              "              "

0710  D3  SEP  R3 ;Do MLS -- set RA.0 = board address per VA VB
  11  F8  LDI
  12  01
  13  5A  STR  RA ;Store an 01 white piece @ VA VB (piece moving)
  14  86  GLO  R6 ;D=R6.0 which is = FA from refer sub (VA)
  15  A7  PLO  R7 ;R7.0=R6.0
  16  17  INC  R7 ;R7 now points to Chip-8 VB
  17  06  LDN  R6 ;D=M(R(6))  Get value of VA
  18  BD  PHI  RD ;Put in RD.1 to save original VA
  19  07  LDN  R7 ;D=M(R(7))  Get value of VB
  1A  AD  PLO  RD ;Put in RD.0 to save original VB
  1B  F8  LDI      ;Set RF.0 = FF = cycle index #2 (VY)
  1C  FF           ;    "              "              "
  1D  AF  PLO  RF  ;    "              "              "
  1E  F8  LDI      ;Set RE.0 = FF = cycle index #1 (VX)
  1F  FF           ;    "              "              "

0720  AE  PLO  RE  ;    "              "              "
  21  F8  LDI      ;Set R3.0 = address search sub
  22  24
  23  A3  PLO  R3
  24  D3  SEP  R3 ;Do MLS -- add indexes to VA VB and set "I" to
  25  0A  LDN  RA ;Get piece @ M(R(A))                board square
  26  FB  XRI      ;Test if = 01 white
  27  01
  28  32  BZ       ;If so, done search, branch to 0740 to change line
  29  40
  2A  0A  LDN  RA ;Get same piece
  2B  FB  XRI      ;Test if = 80 black
  2C  80
  2D  32  BZ       ;If so, branch to 0721 to continue search
  2E  21
  2F  9D  GHI  RD ;Reset VA value

0730  56  STR  R6  ;    @ M(R(6))
  31  8D  GLO  RD ;Reset VB value
  32  57  STR  R7  ;    @ M(R(7))
  33  1E  INC  RE ;Add 01 to RE index #1
  34  8E  GLO  RE
  35  FB  XRI      ;Test if RE.0 = 02 yet
  36  02
  37  3A  BNZ      ;If not, branch to 0721
```

```
0738  21              ;     to do another line
  39  1F  INC   RF   ;Add 01 to RF index #2
  3A  8F  GLO   RF
  3B  FB  XRI        ;Test if RF.0 = 02 yet
  3C  02
  3D  3A  BNZ        ;If not, branch to 071E
  3E  1E             ;     to do another 3 lines
  3F  DC  SEP   RC   ;Return (via sub handler)
```

(CHANGE LINE)

```
0740  9D  GHI   RD ;Reset VA value
  41  56  STR   R6 ;  "        "
  42  8D  GLO   RD ;Reset VB value
  43  57  STR   R7 ;  "        "
  44  F8  LDI       ;R3.0 = address search sub
  45  24
  46  A3  PLO   R3
  47  D3  SEP   R3 ;Do MLS -- add indexes + VA VB/Set "I" to
  48  0A  LDN   RA ;Get piece @ RA                board square
  49  F6  SHR       ;Shift LSB into DF to check if = 01 white piece
  4A  33  BDF       ;If so, branch to 072F
  4B  2F            ;     Change complete
  4C  F8  LDI       ;Else, store white in board (where
  4D  01            ;     there was a black)
  4E  5A  STR   RA
  4F  30  BR        ;Loop till change is complete

0750  44            ;     to 0744
  51  00  FILLER
```

MOVE GENERATOR SUB

```
0752  GENER :6E00 ;VE=00 -- Set VE=00 = move list index
  54         6A01 ;VA=01 -- VA is VX board square     Set to
  56         6B01 ;VB=01 -- VB is VY   "       "        square #1
  58  GEN1   :80A0 ;V0=VA -- Save VA VB in V0 V1
  5A         81B0 ;V1=VB --   "          "
  5C         0864 ;LEGAL -- Do MLS -- test for legal move @ VA VB
  5E         3F00 ;SK=00 -- If VF flag = 00 = legal, skip into next

0760         176A ;GEN2  -- Else jump -- do not store illegal move
  62         AA64 ;LIST  -- Set "I" to move list
```

```
0764          FE1E ;I+VE  -- Add VE index for next list entry
  66          F155 ;PUT   -- Store V0:V1 @ I (I-2)
  68          7E03 ;VE+03 -- Add 3 to VE index for next use
  6A  GEN2  :8A00 ;VA=V0  -- Restore VA value held in V0
  6C          8B10 ;VB=V1 --   "     VB   "     "     V1
  6E          7A01 ;VA+01 -- Add 01 to VA for next in row

0770          3A09 ;SK=09 -- Skip if past eighth square
  72          1758  GEN1  -- Jump to loop a full row
  74          6A01 ;VA=01 -- Reset VA to begin new row
  76          7B01 ;VB+01 -- Add 01 to VB for next row
  78          3B09 ;SK=09 -- Skip if past eighth row
  7A          1758  GEN1  -- Jump to do another row
  7C          AA64  LIST  -- Set "I" to move list
  7E          FE1E ;I+VE  -- Add VE index for next entry

0780          60FF ;V0=FF
  82          F055 ;PUT   -- Store FF end list marker @ M(I)
  84          00EE ;RET   -- Return
```

EVALUATION CONTROLLER SUB

```
0786  EVCNT :6E00 ;VE=00  -- Set VE move list index = 00
  88  EVC1  :AA00 ;TEMP  -- Set "I" to temporary saved board
  8A          096F ;TRANS -- Do MLS -- transfer saved board back
  8C          0064 ; #    -- Number bytes transfer (64=100 decimal)
  8E          0800 ;BOARD -- Address where board goes

0790          AA64 ;LIST  -- Set "I" to move list
  92          FE1E ;I+VE  -- Add index to cycle through all moves
  94          F165 ;GET   -- V0 V1 = move from list
  96          40FF ;SK≠FF -- If V0 ≠ FF, end of list, skip
  98          00EE ;RET   -- Else return, all moves evaluated
  9A          8A00 ;VA=V0 -- Let VA = VX of move
  9C          8B10 ;VB=V1 --  "  VB = VY  "    "
  9E          0700 ;MKMOV -- Do MLS -- make move

07A0          08B1 ;EVAL  -- Do MLS -- evaluate resulting position
  A2          AA64 ;LIST  -- Set "I" to move list
  A4          FE1E ;I+VE  -- Index to move just made
  A6          F165 ;GET   -- This advances "I" to weight slot
  A8          8020 ;V0=V2 -- V0=V2 for storing weight
  AA          F055  PUT   -- Store weight with move
  AC          7E03 ;VE+03 -- Add 3 to move list index
  AE          1788  EVC1  -- Jump to evaluate next move or exit
```

INPUT MOVE

```
07B0  INPUT :2542 ;GETKY -- Do sub -- V0=keypress in 0-8 range or = E
  B2        400C ;SK≠0C -- If not Key C, skip to continue
  B4        00EE ;RET   -- Return, V0 signals player pressed C
  B6        8A00 ;VA=V0 -- Let VA = first keypress
  B8        2542 ;GETKY -- Do sub -- V0=keypress
  BA        400C ;SK≠0C -- If not Key C, skip next
  BC        00EE ;RET   -- Return--Key C pressed (V0=0C)
  BE        8B00 ;VB=V0 -- Let VB = second keypress

07C0        80A0 ;V0=VA -- Save VA VB in V0 V1 (next sub changes VA VB)
  C2        81B0 ;V1=VB --  "                "              "
  C4        0864 ;LEGAL -- Do MLS -- Test for legal move
  C6        3F00 ;SK=00 -- If VF=00, then skip--move is legal
  C8        17D4  ERROR -- Jump to error tone on illegal move
  CA        8A00 ;VA=V0 -- Restore VA VB values held in V0 V1
  CC        8B10 ;VB=V1 --    "              "            "
  CE        26D8 ;DSPMV -- Do sub -- display move

07D0        6000 ;V0=00 -- Signals calling routine a move selected
  D2        00EE ;RET   -- Return (move in VA VB & displayed)
  D4  ERROR :6060 ;V0=60 -- V0=60 for error tone
  D6        F018 ;TONE  -- Sound tone--move illegal
  D8        17B0  INPUT -- Jump to input new move
  DA        0000 --FILLER
```

SCORING

```
07DC  S-1   :0000 ;2BYT  -- Storage for old scores
  DE  SCORE :A7DC  S-1   -- Set "I" to old score above

07E0        F165 ;GET   -- Let V0 V1 = old score
  E2        27E8  OLD   -- Do sub -- erase old score
  E4  SCOR2 :A800 ;BOARD -- Set "I" to computer board (entry #2)
  E6        097E ;COUNT -- Do MLS -- V0=# whites/V1=# blacks
  E8  OLD   :A7DC  S-1   -- Set "I" to old score work area
  EA        F155 ;PUT   -- Store for next call (to erase)
  EC        6C08 ;VC=08 -- VC is VX for display
  EE        6D20 ;VD=20 -- VD  " VY  "     "

07F0        8810 ;V8=V1 -- Save V1 value in V8
  F2        26A2  NUMBS -- Do sub -- convert and display # in V0
  F4        7D07 ;VD+07 -- Add 07 to VY for next number (black)
  F6        8080 ;V0=V8 -- Let V0=V1 value held in V8
  F8        26A2  NUMBS -- Do sub -- convert and display # in V0
  FA        00EE ;RET   -- Return
```

07FC-07FE -- Not used

07FF 00 Byte for level of play storage

0800-0863 -- 100 bytes for computer board

MLS - TEST LEGAL MOVE IN VA VB

```
0864  F8  LDI
  65  09
  66  BC  PHI   RC ;Set RC=address sub handler @ 0918
  67  F8  LDI      ;     to switch to R4 = PC
  68  18
  69  AC  PLO   RC
  6A  DC  SEP   RC ;Do sub--this sub begins running in R4
  6B  F8  LDI      ;Set RA.1=base address computer board
  6C  08           ;     @ 0800
  6D  BA  PHI   RA ;RA.1=0A
  6E  F8  LDI      ;Set R3=address of reference RA sub
  6F  09           ;     for addressing board squares with RA

0870  B3  PHI   R3 ;  "              "              " @ 0901
  71  F8  LDI      ;  "              "                "
  72  01           ;  "              "                "
  73  A3  PLO   R3 ;  "              "                "
  74  D3  SEP   R3 ;Do MLS--set RA per VA VB to board square address
  75  0A  LDN   RA ;Get piece @ RA (@ board square VA VB)
  76  3A  BNZ      ;If ≠ 00, square is not empty, branch
  77  A5           ;     to 08A5 to set illegal move flag & return
  78  86  GLO   R6 ;Get R6.0 which was set to address VA by refer sub
  79  A7  PLO   R7 ;Put in R7.0
  7A  17  INC   R7 ;Increment R7 to point to VB index (VY)
  7B  06  LDN   R6 ;Get VX index @ M(R(6))
  7C  FF  SMI      ;Subtract 01
  7D  01
  7E  56  STR   R6 ;And return to M(R(6))
  7F  07  LDN   R7 ;Get VY index @ M(R(7))

0880  FF  SMI      ;Subtract 01
  81  01
  82  57  STR   R7 ;And return to M(R(7))
  83  F8  LDI      ;Set RE.0=3=loop count #1 (main)
  84  03
  85  AE  PLO   RE
  86  F8  LDI      ;Set RF.0=3=loop count #2 (secondary)
  87  03
  88  AF  PLO   RF
```

This sets up start point @ VX-1; VY-1

```
0889  D3  SEP   R3 ;Do MLS--set RA to board square @ VA VB
  8A  0A  LDN   RA ;Get piece @ RA
  8B  32  BZ       ;If=00 (square empty) branch to continue search
  8C  91
  8D  FB  XRI      ;Test if=FF border byte
  8E  FF
  8F  3A  BNZ      ;If not, branch to 08AA to set legal move flag

0890  AA           ;     (at least one adjacent square occupied)
  91  06  LDN   R6 ;Get VX @ M(R(6))
  92  FC  ADI      ;Add 01 for next in row
  93  01
  94  56  STR   R6 ;And replace @ M(R(6))
  95  2F  DEC   RF ;Loop count #2 - 01
  96  8F  GLO   RF
  97  3A  BNZ      ;Loop until 3 squares tested in tow
  98  89
  99  06  LDN   R6 ;Get VX @ M(R(6))
  9A  FF  SMI      ;Subtract 03 to reset to row beginning
  9B  03
  9C  56  STR   R6 ;And replace @ M(R(6))
  9D  07  LDN   R7 ;Get VY @ M(R(7))
  9E  FC  ADI      ;Add 01 for next row
  9F  01

08A0  57  STR   R7 ;And replace @ M(R(7))
  A1  2E  DEC   RE ;Loop count #1 - 01
  A2  8E  GLO   RE
  A3  3A  BNZ      ;Loop (next row) till 3 rows tested
  A4  86           ;     if necessary
  A5  F8  LDI      ;Set R6 to point to Chip-8 VF (flag)
  A6  FF           ;   "            "              "
  A7  A6  PLO   R6 ;   "            "              "
  A8  56  STR   R6 ;Store FF illegal flag as Chip-8 VF
  A9  DC  SEP   RC ;Return (via sub handler)
  AA  F8  LDI      ;Set R6 to point to Chip-8 VF (flag)
  AB  FF           ;  "              "              "
  AC  A6  PLO   R6 ;  "              "              "
  AD  F8  LDI      ;Get 00 byte
  AE  00
  AF  56  STR   R6 ;And store as VF to flag legal move
08B0  DC  SEP   RC ;Return (via sub handler)
```

MLS - BOARD EVALUATION

```
08B1  F8  LDI      ;Set RC=address sub handler @ 0918
  B2  09           ;  "                "              "
  B3  BC  PHI  RC  ;  "                "              "
  B4  F8  LDI      ;  "                "              "
  B5  18           ;  "                "              "
  B6  AC  PLO  RC
  B7  DC  SEP  RC  ;Call sub--switch to PC=R4
  B8  F8  LDI      ;Set R3=address count sub @ 097E
  B9  09           ;  "                "              "
  BA  B3  PHI  R3  ;  "                "              "
  BB  F8  LDI      ;  "                "              "
  BC  7E           ;  "                "              "
  BD  A3  PLO  R3  ;  "                "              "
  BE  F8  LDI      ;Set RA=address of board @ 0800
  BF  08           ;  "                "          "

08C0  BA  PHI  RA  ;  "                "          "
  C1  F8  LDI      ;  "                "          "
  C2  00           ;  "                "          "
  C3  AA  PLO  RA  ;  "                "          "
  C4  D3  SEP  R3  ;Call sub--count pieces
  C5  E2  SEX  2   ;X=2 on return as last sub changes X
  C6  22  DEC  R2  ;Stack pointer free
  C7  F8  LDI
  C8  80
  C9  52  STR  R2  ;Push 80 (mid weight)
  CA  8E  GLO  RE  ;D=RE.0 (# whites)
  CB  F4  ADD      ;Add # whites to weight on stack
  CC  52  STR  R2  ;Push
  CD 8F/E2 GLO RF  ;D=RF.0 (#blacks)/NOPS--see notes for
  CE F5/E2 SD      ;Subtract #blacks/   strategy test (tape
  CF 52/E2 STR R2  ;Push                supplied with the NOPS)

08D0  94  GHI  R4  ;Set R3 to address adjust sub
  D1  B3  PHI  R3  ;  "            "          "
  D2  F8  LDI      ;  "            "          "
  D3  EC           ;  "            "          "
  D4  A3  PLO  R3  ;  "            "          "
  D5  F8  LDI
  D6  0B
  D7  AA  PLO  RA  ;RA=corner #1
  D8  D3  SEP  R3  ;Do sub--adjust for corners
  D9  F8  LDI
  DA  12
  DB  AA  PLO  RA  ;RA=corner #2
  DC  D3  SEP  R3  ;Do sub--adjust for corners
```

```
08DD  F8  LDI
  DE  51
  DF  AA  PLO   RA ;RA=Corner #3

08E0  D3  SEP   R3 ;Do sub--adjust for corners
  E1  F8  LDI
  E2  58
  E3  AA  PLO   RA ;RA=corner #4
  E4  D3  SEP   R3 ;Do sub--adjust for corners
  E5  F8  LDI
  E6  F2
  E7  A6  PLO   R6
  E8  72  LDXA     ;Pop weight (reset stack pointer)
  E9  56  STR   R6 ;Store as V2
  EA  DC  SEP   RC ;Return via sub handler
```

ADJUST SUB (FOR CORNERS)

```
08EB  D4  SEP   R3 ;Return
  EC  0A  LDN   RA ;Entry--get corner piece
  ED  F6  SHR      ;Shift right to check for white
  EE  3B  BNF      ;If not, branch to exit
  EF  EB

08F0  F8  LDI      ;Else add 5 to weight on stack
  F1  05           ;(change to fine-tune strategy)
  F2  F4  ADD
  F3  52  STR   R2 ;Store new weight on stack
  F4  30  BR       ;Branch to exit
  F5  EB
```

MLS - REFERENCE RA TO BOARD ADDRESS PER VA VB COORDINATES

```
0900  D4  SEP   R4 ;Return-leaving R3 @ entry for other MLS calls
  01  E2  SEX   R2 ;X=2 (entry here)
  02  22  DEC   R2 ;Stack free
  03  F8  LDI      ;Set R6.0=FB
  04  FB
  05  A6  PLO   R6 ;R6.0 points to Chip-8's VB=VY board coordinate
  06  06  LDN   R6 ;D=M(R(6))--get VY value
  07  FE  SHL      ;Shift to multiply x 2
  08  52  STR   R2 ;Push
  09  FE  SHL      ;Further multiply x 2
```

```
090A  FE  SHL      ;  "        "          " (total x 8 here)
  0B  F4  ADD      ;Add (x 8 value) + (x 2 value)=(x 10 value total)
  0C  26  DEC  R6  ;R6 points to Chip-8's VA=VX board coordinate
  0D  E6  SEX  6   ;X=6 for next add instruction
  0E  F4  ADD      ;Add (x 10 VY value)+(VX value)=address
  0F  AA  PLO  RA  ;RA.0=D--put result in RA to address board square

0910  12  INC  R2  ;Reset stack pointer
  11  30  BR       ;Branch to exit @ 0900
  12  00
```

0913-0917 -- Not used -- FILLER

SUB HANDLER (FOR RUNNING ROUTINES IN R4)

```
0918  93  GHI  R3  ;D=R3.1
  19  B4  PHI  R4  ;R4.1=R3.1
  1A  83  GLO  R3  ;D=R3.0
  1B  A4  PLO  R4  ;R4.0=R3.0--set R4=return address to sub
  1C  D4  SEP  R4  ;Calling sub runs in R4
  1D  F8  LDI      ;Handler's PC points to here
  1E  00
  1F  B4  PHI  R4  ;Set R4=0042 for return to Chip-8

0920  F8  LDI      ;    Interpreter control
  21  42           ;   "       "       "       "
  22  A4  PLO  R4  ;   "       "       "       "
  23  D4  SEP  R4  ;Return control to Chip-8 Interpreter
```

MLS - SEARCH SUB

```
0924  E2  SEX  2   ;X=2
  25  22  DEC  R2  ;Stack pointer free
  26  8E  GLO  RE  ;D=Index #1
  27  52  STR  R2  ;Push for adding
  28  06  LDN  R6  ;D=M(R(6)) = VA = VX
  29  F4  ADD      ;Add to index on stack
  2A  56  STR  R6  ;Return to M(R(6)) = VA
  2B  8F  GLO  RF  ;D=Index #2
  2C  52  STR  R2  ;Push for adding
  2D  07  LDN  R7  ;D=M(R(7)) = VB = VY
  2E  F4  ADD      ;Add to index on stack
  2F  57  STR  R7  ;Return to M(R(7)) = VB

0930  30  BR       ;Branch to refer entry to conclude sub
  31  03           ;     @ 0903 (stack reset there too)
```

MLS - COMPUTER 8x8 GRID SET-UP

```
0932  22  DEC   R2 ;Stack pointer free
  33  8A  GLO   RA ;RA="I"=base address computer board (set by caller)
  34  FC  ADI      ;Add 63 hex to RA.0 to find last address of board
  35  63
  36  52  STR   R2 ;Push RA.0 + 63 hex
  37  F8  LDI      ;D=FF byte for board's boarders
  38  FF
  39  5A  STR   RA ;Store FF @ M(R(A))
  3A  8A  GLO   RA ;D=RA.0 to test for done
  3B  1A  INC   RA ;RA+1 for next byte (if not done)
  3C  F3  XOR      ;Compare RA.0;byte on stack
  3D  3A  BNZ      ;If ≠ yet, loop back to continue, else
  3E  37           ;     fall through to next part
  3F  EA  SEX   RA

0940  8A  GLO   RA ;D=RA.0
  41  FF  SMI      ;Subtract 0A from RA.0 address to prepare
  42  0A           ;for next part
  43  AA  PLO   RA ;Return adjusted address to RA.0
  44  F8  LDI
  45  08
  46  AC  PLO   RC ;Set RC.0=08=loop count #1 (main)
  47  2A  DEC   RA ;RA=RA-02 to skip over FF border bytes
  48  2A  DEC   RA
  49  F8  LDI
  4A  08
  4B  AD  PLO   RD ;Set RD.0=08=loop count #2 (secondary)
  4C  94  GHI   R4 ;(=00 as R4 always address 0042 at this point)
  4D  73  STXD     ;Store @ M(R(X))=RA and decrement RA by 1
  4E  2D  DEC   RD ;Decrement loop count #2
  4F  8D  GLO   RD ;D=RD.0 = loop count #2

0950  3A  BNZ      ;If ≠ 00, branch up to store 8 00's total
  51  4C
  52  2C  DEC   RC ;Decrement loop count #1
  53  8C  GLO   RC ;D=RC.0 = loop count #1
  54  3A  BNZ      ;If ≠ 00, branch up to store 8 oo's total
  55  47
  56  12  INC   R2 ;Reset stack pointer
  57  D4  SEP   R4 ;Return control to Chip-8 Interpreter
```

MLS - BEGINNING POSITION-NEW GAMES

```
0958  F8  LDI      ;Set a white piece in RF.1
  59  01           ;  "          "          "
  5A  BF  PHI  RF  ;  "          "          "
  5B  F8  LDI      ;Set a black piece in RF.0
  5C  80           ;  "          "          "
  5D  AF  PLO  RF  ;  "          "          "
  5E  F8  LDI      ;Set RA to square 4:4
  5F  2C

0960  AA  PLO  RA
  61  9F  GHI  RF ;Get white
  62  5A  STR  RA ;Store @ 4:4
  63  1A  INC  RA ;Set RA to square 5:4
  64  8F  GLO  RF ;Get black
  65  5A  STR  RA ;Store @ 4:5
  66  F8  LDI      ;Set RA to square 4:5
  67  36
  68  AA  PLO  RA
  69  8F  GLO  RF ;Get black
  6A  5A  STR  RA ;Store @ 4:5
  6B  1A  INC  RA ;Set RA to square 5:5
  6C  9F  GHI  RF ;Get white
  6D  5A  STR  RA ;Store @ 5:5
  6E  D4  SEP  R4 ;Return control to Chip-8 Interpreter
```

MLS - TRANSFER

```
096F  45  LDA  R5 ;Advance R5 to # bytes

0970  45  LDA  R5 ;Get # bytes for transfer from caller
  71  AD  PLO  RD ;Put in RD.0 as loop count
  72  45  LDA  R5 ;Get to address  (high byte)
  73  BC  PHI  RC ;Put in RC.1
  74  45  LDA  R5 ;Get to address  (low byte)
  75  AC  PLO  RC ;Put in RC.0
  76  4A  LDA  RA ;Get one byte for transfer @ M(R(A))
  77  5C  STR  RC ;Store at new location @ M(R(C))
  78  1C  INC  RC ;Increment new location pointer RC
  79  2D  DEC  RD ;Loop count - 01
  7A  8D  GLO  RD ;Get the loop count in RD.0
  7B  3A  BNZ      ;If ≠ 00 yet, loop to continue transfer
  7C  76
  7D  D4  SEP  R4 ;Else return control to Chip-8 Interpreter
```

MLS - COUNT WHITES/BLACKS

```
097E  22  DEC   R2 ;Stack pointer free
  7F  8A  GLO   RA ;D=RA.0 for adjusting "I" pointer

0980  52  STR   R2 ;Push RA.0 for later done check
  81  FC  ADI      ;Add 59 hex to RA.0
  82  59
  83  AA  PLO   RA ;Put back in RA.0--points to last board square
  84  88  GLO   R8 ;(=00 as R8.0 is tone duration which is off here)
  85  AE  PLO   RE ;RE.0=00=white pieces count
  86  AF  PLO   RF ;RF.0=00=black pieces count
  87  0A  LDN   RA ;Get byte from board @ M(R(A))
  88  FB  XRI      ;Test if = FF border byte
  89  FF
  8A  32  BZ       ;If so, branch to skip count next
  8B  95
  8C  0A  LDN   RA ;Get same board byte (either 00, 80 or 01)
  8D  32  BZ       ;If=00, then empty--branch past next part
  8E  95
  8F  F6  SHR      ;Shift LSB into DF register

0990  3B  BNF      ;Branch on DF=0 (piece is 80 hex=black)
  91  94
  92  1E  INC   RC ;RC+1 counts the white piece
  93  38  SKP      ;Skip next instruction always
  94  1F  INC   RD ;RD+1 counts the black piece
  95  2A  DEC   RA ;RA-1 addresses next board square
  96  8A  GLO   RA ;D=RA.0 (get address RA.0 to test if done)
  97  F3  XOR      ;Compare RA.0:byte on stack (original RA.0)
  98  3A  BNZ      ;If ≠, branch to continue count
  99  87
  9A  F8  LDI      ;Set R6=address Chip-8's V0
  9B  F0
  9C  A6  PLO   R6
  9D  8E  GLO   RE ;D=RE.0=number white pieces
  9E  56  STR   R6 ;M(R(6))=D -- store as V0 value
  9F  16  INC   R6 ;R6+1 points to Chip-8's V1

09A0  8F  GLO   RF ;D=RF.0=number black pieces
  A1  56  STR   R6 ;M(R(6))=D -- store as V1 value
  A2  12  INC   R2 ;Reset stack pointer
  A3  D4  SEP   R4 ;Return
```

MLS - FLIP FLOP BOARD

```
09A4  22  DEC   R2 ;Stack pointer free
  A5  8A  GLO   RA ;D=M(R(A))
  A6  52  STR   R2 ;Push RA.0 for later done test
  A7  FC  ADI      ;Add 58 hex to RA.0 to address last board square
  A8  58
  A9  AA  PLO   RA ;RA.0=D (place new address in RA.0)
  AA  0A  LDN   RA ;D=M(R(A)) -- get a byte from board
  AB  32  BZ       ;If = 00, empty space, branch past next part
  AC  B5           ;      which compliments pieces
  AD  FB  XRI      ;Test if = FF (border byte)/pieces
  AE  FF
  AF  32  BZ       ;If = FF, branch to skip next

09B0  B5
  B1  0A  LDN   RA ;D=M(R(A)) -- get byte from board (piece)
  B2  FB  XRI      ;Exclusive "OR" with 81 to compliment
  B3  81
  B4  5A  STR   RA ;Replace complimented byte in board
  B5  2A  DEC   RA ;RA points back to next byte in board
  B6  8A  GLO   RA ;D=RA.0 to test for done
  B7  F3  XOR      ;Compare RA.0:byte on stack (original RA.0)
  B8  3A  BNZ      ;If ≠, branch to continue flip flop
  B9  AA
  BA  12  INC   R2 ;Else reset stack pointer
  BB  D4  SEP   R4 ;Return control to Chip-8 Interpreter
```

MLS - GET BEST MOVE

```
09BC  0A  LDN   RA ;Get first byte from move table
  BD  FB  XRI      ;Test if = FF (no moves in list)
  BE  FF
  BF  3A  BNZ      ;If ≠ FF, skip early return

09C0  C2
  C1  D4  SEP   R4 ;Early return--game (or look ahead)over now
  C2  22  DEC   R2 ;Stack pointer free
  C3  1A  INC   RA ;RA+2 points to first weight in list
  C4  1A  INC   RA ;   "          "          "
  C5  9A  GHI   RA ;Set RE=RA
  C6  BE  PHI   RE ;   "    "
  C7  8A  GLO   RA ;   "    "
  C8  AE  PLO   RE ;   "    "
  C9  9E  GHI   RE ;Set RA=RE (needed on subsequent loops)
```

```
09CA  BA  PHI  RA ;   "          "          "
  CB  8E  GLO  RE ;   "          "          "
  CC  AA  PLO  RA ;   "          "          "
  CD  0A  LDN  RA ;Get weight @ RA
  CE  52  STR  R2 ;Push for comparing with others
  CF  1E  INC  RE ;RE+3 points to next weight

09D0  0E  INC  RE ;   "          "          "
  D1  FB  INC  RE ;   "          "          "
  D2  FF  LDN  RE ;Get weight
  D3  32  XRI     ;Check for possible stop byte (end list)
  D4  DD
  D5  1E  BZ      ;If = FF, branch to exit
  D6  1E
  D7  0E  LDN  RE ;Get same weight
  D8  F5  SD      ;Subtract M(R(X))-D = M(R(A))-M(R(E))
  D9  33  BPZ     ;If positive or zero, M(R(A)) ≥ M(R(E)) --
  DA  CF          ;     a greater weight was not found
  DB  30  BR      ;Else set RA=RE and continue
  DC  C9          ;     weight was greater
  DD  2A  DEC  RA ;Set RA ("I") to point to move with
  DE  2A  DEC  RA ;     highest weight
  DF  12  INC  R2 ;Reset stack pointer
09E0  D4  SEP  R4 ;Return control to Chip-8 Interpreter
```

09E1-09FF -- Not used

0A00-0A63 -- Temp store -- intermediate boards are stored here during the look ahead

0A64-0AFF -- Move list #1 -- moves generated are always placed here and the list is a variable size

0B00-0B63 -- Perm store -- computer boards (flipped) are stored here for resetting after look ahead and at the beginning of a look ahead search

0B64-0BFF -- Move list #2 -- primary move list, each entry of which begins a look ahead process

0C00-0DFF -- Character set (modified as follows)

ASCII MESSAGES
(In place of the lower case bit patterns of the character set)

```
0D6C  41  52  45  53  43  4F  20  50  "ARESCO PRESENTS"
0D74  52  45  53  45  4E  54  53  00
```

```
0D7C  56 20 49 20 50 20 2D 20  "VIP-FLOP"
0D84  46 20 4C 20 4F 20 50 00
0D8C  20 20 20 20 20 42 20 59  "BY"
0D94  00 20 20 20 54 4F 4D 20  "TOM SWAN"
0D9C  53 57 41 4E 00 4C 45 56  "LEVEL OF PLAY?"
0DA4  45 4C 20 4F 46 20 50 4C
0DAC  41 59 3F 00 28 4B 45 59  "(KEYS 0-F)"
0DB4  53 20 30 2D 46 29 20 00
0DBC  00 00 56 45 52 53 49 4F  "VERSION?"
0DC4  4E 3F 00 4B 45 59 20 31  "KEY 1 H-C"
0DCC  2D 20 48 2D 43 00 4B 45  "KEY 2 C-C"
0DD4  59 20 32 2D 20 43 2D 43
0DDC  00 4B 45 59 20 33 2D 20  "KEY 3 H-H"
0DE4  48 2D 48 00 57 48 49 54  "WHITE"
0DEC  45 00 42 4C 41 43 4B 00  "BLACK"
0DF4  57 48 49 54 45 00 4D 4F  "WHITE"  *
0DFC  56 45 53 00  -  -  -  -  "MOVES"
```

* NOTE - "WHITE" is repeated for ease in switching the turn indicator form "WHITE" to "BLACK" to "WHITE MOVES."

APPENDIX -- SOME NOTES

The Chip-8 Interpreter supplied with VIP-OKER and VIP-FLOP in this book differs somewhat from the interpreter that comes in the VIP manual for your computer. First of all it takes up more room, three pages instead of two, and second, it is designed to support a two-page high-resolution display rather than the "normal" one-page format usually used for Chip-8 games. The modifications which allow this were detailed in the September 1978 issue of VIPER, the newletter published by ARESCO exclusively for Cosmac 1802 system owners. Some additional changes were added by me and these were described in my book Pips for Vips, also available through ARESCO.

You may use the interpreter for your own games, and in fact you must use it if you decide to adapt some of the subroutines here for your own programming. Other than the two-page display capability, there are five other important differences.

1)The Chip-8 memory map which your manual charts out for your reference and to aid in debugging programs, is unchanged in format, but is located <u>in front</u> of your programs instead of behind them as it was before. All work spaces, the stack, and Chip-8 variables are now

located in the memory space 02A0-02FF corresponding to the chart in your manual.

2)The erase screen instruction, 00E0, must not be used in your programs. Instead, the new instruction 0230 will clear the display though now two pages are set to zeroes instead of only one.

3)All Chip-8 programming must begin at 0300, not at 0200 as for one-page display programs.

4)An instruction now exists, actually it is not a "pure" Chip-8 instruction but rather a machine language subroutine (MLS), that permits multiple keypresses to be combined into one byte. The instruction takes the place of the BMMM instruction in your one-page interpreter, and the BMMM will no longer work. What this does is allow you to press Keys 1 then 2 and the resulting byte will equal the binary equivalent to the decimal number 12. Numbers such as 127 or 216 may be entered from the keyboard and stored in a single byte.

To use the new instruction to combine keypresses, you must first set equal to zero any variable into which you want the input to go. If V1 is the intended variable, you would program

6100;V1=00 -- Let the variable V1=00

Next, program the following sequence:

```
CMBIN :F00A ;V0=KY -- Let V0=value of next keypress
       400E ;SK≠0E -- If not Key E, skip next
       1XXX  EXIT  -- Jump to exit this loop
       01A4 ;MLS   -- Do MLS--combine keypresses
       F1F0 ;V1=V0 -- V1 equals combined values in V0
       1XXX  CMBIN -- Jump back for next key entry
 EXIT :   (Program continues here)
```

An example of the new 01A4 instruction may be seen at location 0C96-0CA0 in the program listing of VIP-OKER. The F1F0 that follows the 01A4 instruction tells the MLS which Chip-8 variables hold the last keypress and which is to get the answer or the combination of all keypresses. The F1 means V1 will receive the answer, and the F0 part tells the routine that V0 holds the most recent keypress value. If you programmed F2F4, for example, V2 would receive the answer and V4 would be the amount for combining. In the example above, pressing Key E will exit the routine with the answer in V1, though any key may be programmed to do this.

You are not limited to combining only keypresses, however. Any string of digits (up to 255) may be combined into a single binary byte if you wish. Perhaps a good way to understand this instruction is to think of it as the reverse of the FX33, convert VX into its three decimal digit equivalent. The 01A4/FXFY instruction pair

takes one, two or three decimal digits, one by one in VY, and combines them into their binary equivalent in VX as long as VX was first set equal to zero.

5)Your new interpreter also provides a special routine used to display text in your programs. No longer must you figure out bit patterns to write words on the screen -- your interpreter does it all for you!

The tape supplied with these games contains a full ASCII character set which takes up two pages of memory at 0C00-0DFF, just in front of the two-page display refresh area. Except for a few minor punctuation symbols, you may use all of the ASCII characters including lower case to write prompts, error messages, and player's names in your programs. But, you must store the character set at 0C00-0DFF in order to make use of the feature.

The instruction that allows this is used in the following manner:

```
        AXXX ;MSG-1 -- Set "I" pointer to message #1
PRINT :0244 ;MLS   -- Do MLS--Call MESSAGER
        DXY5 ;SHOW  -- Display message @ VX VY
```

The "I" pointer is set to any area of memory containing an ASCII string (sequence) up to 16 characters in length and ending with a 00 null character. The

following string would be programmed to print the word VIPER:

56 49 50 45 52 00 "VIPER"

Setting "I" to the address of the first byte of this string, and then programming the above instructions will print the word VIPER starting at the VX VY coordinates you have specified. Messages may be erased by redisplaying the same message twice or, as in VIP-OKER, writing a machine language subroutine to clear a "window" in the display to prepare for the next message.

(A detailed description of the Chip-8 Message and the format for the character set is described in the author's book, Pips for Vips.)

* * *

Each program is supplied with a list of assignments of variables or registers, a memory map and a list of all the subroutines and their addresses. Following each sub may be a one to five character label in parenthesis, for example (SHIMR), which will serve to identify that subroutine in the program listing. Machine language subs are marked "MLS" and may or may not have an associated label. Data areas are usually labeled in the same manner

though not always.

The Chip-8 format differs from the form usually used in printing Chip-8 programs. Please turn to any page containing a Chip-8 routine. You will see the address on the left side of the page denoting the location in memory. Next is a column, for the most part blank, but occasionally containing a short capitalized word, called a label, that marks certain positions in the program. The third column contains the Chip-8 instructions -- nothing different here. Each instruction may or may not be followed by a semicolon which in turn is followed by another short capitalized label (or argument) or a comment. This column will be very useful in following the routines printed here. It may contain a short form description of the previous instruction such as 6101 ;V1=01 which, to the programmer already familiar with Chip-8, will permit fast deciphering of a routine. The short form column may also contain arguments next to subroutine calls which are identical in form to the labels used in column two, the usually empty one. The program flow may be much more easily followed by using the labels and arguments to trace a jump to instruction or a subroutine call instead of working only with addresses not easily remembered. Thus when the program says

"Jump to BLIN1," you may find that location in column two faster than if the program only stated "Jump to 0342." The last column of each Chip-8 listing is a long form description that will further clarify what the program is doing at that point.

Those of you who are familiar with the format of my program "Chip-8 Assembler-3" will appreciate the use of columns two, three and four in the listing. By using a text editor to prepare source listings, and being careful to follow the format instructions for the assembler, routines from the programs here may be taped for assembling and using in your own programs without regard to their location in memory! In any case, the new Chip-8 format will allow you to take in the program flow more easily than before greatly aiding in your understanding of a section's purpose.

If you do prepare source tapes for assembling with Chip-8 Assembler-3, please be careful to include subroutines that are called <u>from</u> the one you are adapting to your own library. The Assembler's built-in error checks will usually let you know if you forget to do this, however, but a little care will go a long way.

* * *

The program descriptions in this book frequently suggest improvements you may want to write to customize your games and sharpen your computer skills. If you own extra memory above the 4K in your Cosmac VIP, you have the opportunity to create super games for your computer. As Chip-8 programs cannot run directly in memory space above 4K, I will assume that you will be working in machine language. This is not a complete solution to the problem as machine language subs cannot be directly called above 4K from Chip-8 routines. There are ways around this, though, and here is one suggestion.

A machine language sub above 4K could run using a register other than R3 as the program counter. You will need a machine language controller which you first call from your Chip-8 routine. The controller resides in a memory area <u>below</u> the 4K boundary at 0FFF, and it's function is to call the proper routine above the 4K line. The address of the subroutine you want to call above 4K immediately follows the call to the machine language sub. It is said that the "caller passes the address" to the controller sub. For example:

```
0MMM ;CONT -- Do MLS--Call sub controller
1692 ;SUB  -- Address of MLS @ 1692
```

The controller in turn will call the sub at 1692. Here's how this may be accomplished.

CONTROLLER

```
LDA  R5 -- Get high byte sub address "passed by caller"
PHI  RC -- Place in RC.1
LDA  R5 -- Get low byte sub address "passed by caller"
PLO  RC -- Place in RC.0
SEP  RC -- Call sub     (end controller)
```

The sub at 1692 will run using RC as the program counter. Any other register could be used with respect for the scheme of the Chip-8 register assignment. R3 will be free for use in your sub above the 4K line. Returning control to the Chip-8 interpreter is done in the usual way with a D4 SEP R4 -- Return instruction.

The above suggestion is small enough that you should be able to find a space for it somewhere in your interpreter as a permanent addition for using your extra memory in Chip-8 programs.

TAPE ORGANIZATION

(All programs are loaded from 0000)

Location	# pages	Description
0000	-	30 second leader
0013	-	1 minute signal for level check
	-	30 second leader
0050	14 (E)	VIP-OKER
	-	30 second leader
0079	14 (E)	VIP-FLOP
	-	30 second leader
0105	14 (E)	VIP-OKER (attract mode)
	-	30 second leader
0132	2 (@ 0C00)	ASCII character set
	-	
	-	One minute separation
0158	(Above repeated minus the 1 minute signal test)	

VIP HARDWARE AND SOFTWARE

The prices here are believed to be accurate and effective as of Dec. 1, 1979. You may order RCA produced items from ARESCO, as well as the VIP software. Be sure to write on you order for any RCA products that you are aware that delivery may take more than 30 days.

RCA PRODUCED PRODUCTS

VP44	Four 2114 RAMs for on-board expansion	$ 36.00
VP45	Four 9131 RAMs for on-board expansion	36.00
VP311	VIP Instruction Manual	5.00
VP320	VIP User Guide	5.00
VP550	Supersound Board	49.00
VP560	EPROM Board	34.00
VP565	EPROM Programmer Board	99.00
VP570	Memory Expansion (4K static RAM)	95.00
VP575	System Expansion	59.00
VP580	Expansion Keypad	15.00
VP585	Keypad Interface Board	10.00
VP590	Color Board	69.00
VP595	Simple Sound Board	24.00
VP601	ASCII Keyboard	65.00
VP700	Tiny BASIC ROM Board	39.00
VP710	VIP Game Manual.....(20 games)	10.00
VP711	VIP (Assembled)	199.00
TC1210	9" Video Monitor	195.00
TC1212	12" Video Monitor	325.00
TC1217	17" Video Monitor	435.00
CDP18S731	Four 9131 RAMs plus necessary components for I/O expansion ports. Used only with the kits.	69.00
CDP18S745	Four 2114 RAMs plus necessary components for I/O expansion ports. Used only with the kits.	69.00
MPM201B	CDP 1802 User Manual	5.00
VP620	ASCII Keyboard Connector cable	20.00

Sam Hersh Editor.(cassette only)	5.00
Don Stein Editor (with documentation)	15.00
Don Stein Editor (cassette only)	5.00
Brian Astle's LIFE (with documentation)	10.00
Clarke Hottle's BASEBALL (cassette only)	10.00
Udo Pernisz's LUNAR LANDER (cassette only)	5.00
Robert Rupp's BLACKJACK.(cassette only)	5.00
Stanley Bayless's TREASURE HUNT (cassette only)	5.00
Robert Rupp's CRAPS (cassette only)	5.00
Tom Swan's PIPS FOR VIPS I	19.95
Tom Swan's PIPS FOR VIPS II	19.95

We also have the 1861 data sheet (for $1.50). Please send an additional $1.00 for postage and handling if you order an RCA item. RCA charges us postage from their plant to our house, and we prepay shipping from our house to yours. Your dollar will help defray the costs of all that!

ARESCO
P. O. Box 1142
Columbia, MD 21044

Before ordering products from ARESCO, please read this notice!

We, like everyone else, must wait in line for RCA to deliver product to us. We have an advantage over the individual VIP owner, however, in that we can telephone a purchase order number and pay for the product when it arrives rather than by prepayment. We do not cash your check, nor charge your credit card, until the day we ship your merchandise to you. So you will have the use of those funds, rather than RCA or ARESCO, until then. Secondly, we do not, as a rule, stock any items listed as available from RCA. We don't have the capital to handle that. Therefore, you must make note on any order for RCA products that you are aware that shipment may take more than 30 days. It usually does. And, finally, we cannot accept purchase orders ourselves. We have found that purchase orders from even the most reputable firms do not get paid in time to meet our needs, and we don't have the capital to handle that, either.

To order any product listed simply fill in the blanks below. Please give a street address for UPS delivery within the USA, or send $4.00 for delivery by US mail to your post office box. For subscription orders, non USA people should send an extra $10 for the volume, if they want air mail. If not, it will go out second-class, like the USA subscriptions do.

Please ship the following items. I understand that payment in full must accompany my order, but that you will not charge my credit card or cash my check until the day you ship my products to me.

NAME_____

ADDRESS_____

CITY:_____STATE:_____ZIP:_____

MC/VISA #_____EXP DATE:_____
(If you're using Master Charge, we need the "other" four numbers on your card. These digits represent the Interbank Number. Visa cards don't have such a number.) INTERBANK #_____

ITEMS ORDERED:
VIPER VOL. 1 (+ Index) - $15.00 _____
VIPER VOL. 2 - $15.00_____
PIPS FOR VIPS(+tape) - $19.95 _____
PIPS FOR VIPS (no tape)- $14.95 _____

ARESCO SOFTWARE:_____

_____

_____

RCA PRODUCTS:_____

_____

_____

Credit card holder's signature:_____

Total amount authorized or enclosed: $_____

* Initial here to indicate you understand that delivery of RCA products may take more than 30 days._____