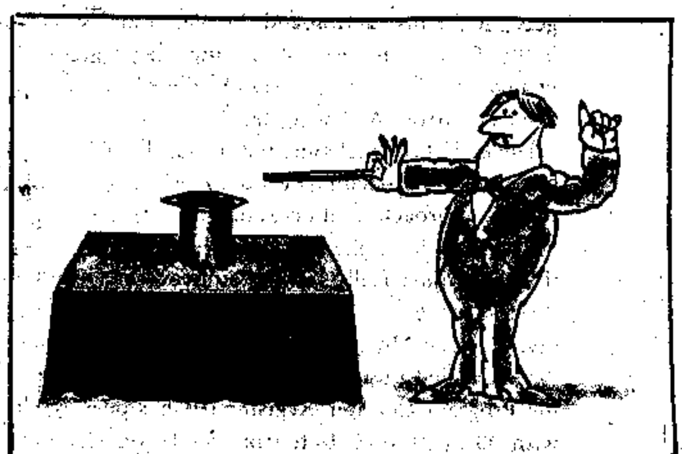
Questdata

Volume 1 · Issue #9 ©

on underlying structures

Beauty Beneath the Machine Language Surface

He was about to trap the robot's queen, a suspenseful move because this robot was programmed for eccentricity; and at a given moment it could be functioning on any level from idiot to genius. He never knew whether an apparent lapse was due to stupidity or the setting of a cunningly contrived trap.
—*from* MONUMENT *by Lloyd Biggle, Jr.*



INGENUITY COUNTS. The design of the 1802 microprocessor leads you along the path toward creative insight into basic structures or algorithms. It is perhaps the lack of MOVE BLOCK type instructions which reveals the true nature and elegance of the underlying logical patterns and structures. There is a whole world of logical relationships which are missed as things become "easier for the user." There are three or possibly more (see Machine Language section of QUESTDATA No. 5) ways to move a block of material. There is a kind of philosophical or personal choice involved here. Do you want convenience food (TV dinners, canned spaghetti) or do you want to cook from scratch?

Actually, having once constructed your clever MOVE BLOCK or whatever routine, you can package it as a subroutine and forget about it. Even your CALL and RETURN structures are not automatic functions with the COSMAC. With other computers you simply punch in CD (hex code for Z80 CALL) and you are off to the races. But you miss out on the elegance of the race. There is more than aesthetics at stake for unsound structures will often result in programs which take up unnecessary amounts of memory and run at slower speeds. There is a price to be paid for the logical beauty and cleverness of the COSMAC. The price is in time spent learning the true nature of things.

Let's take the 1802 JUMP instruction. The true nature of a jump to another location is that the next thing the Program Counter (PC) reads is the next location the computer fetches and executes. The PC register, after fetching an instruction, increments itself so that it points to the next location in memory. This bookkeeping ability of incrementing the pointer to the next location is something the programmer really does not have to think about very much. Sure, a computer goes through its memory sequentially, much as a shopper checks off items as they are purchased from a shopping list. A JUMP can be thought of as a GOTO and then an address of where to go. Or you can think of it as a change in where the bookkeeping PC is pointing. This is why the 30 (BR) is listed on the RCA summary as M(R(P)) → R(R).0. That is shorthand for "the byte following the 30 instruction always replaces the low order byte of the Program Counter R(P)." Right, it is easier to think of the 30 (BR) as a GOTO.

With the COSMAC you have a way of verifying that the 30 instruction shoves the "where to" into the low byte of the Program Counter Register. This is very much like a mathematical proof which goes about its proving by taking a different direction to reach the same conclusion. Also, what we are about to do is something you can't possibly do on a Z-80, 8080 or 6800 because you can't touch the PC in those machines in the manner we can with the 1802. The magician rolls up his sleeves, takes out his magic wand, mumbles a few choice words over the hat and then . . .

Try plugging in the classic 7B (turn on Q), 7B, 7B, 7B, 7B, 7B, 7B, 7B, 7A, 7A, 7A, 7A, 7A, 7A, 30, 00. It really does not matter how many 7B's and 7A's you put into your microcomputer—just remember that the more of them you put in, the lower your tone will be when you run the program, or the dimmer your Q-LED will be when you run the program. So, OK, load up the classic tone program, starting at location 00 and run. A tone results. But we've seen that trick before, the audience protests. The program gets the 30 instruction and the next address executed is 00. That is one way of viewing the program. Now, change the 30 00 instruction to F8 00 A0 and press the GO button. A tone again.

The audience laughs and points out that the magician has used more machine cycles (four instead of two for the BR approach) and takes one mere instruction.

The magician sighs and gives the following explanation to a quizzically smiling audience . . . "Ummmm, there is more than one way to peel a banana as they say, In effect, by peeling the banana at both ends we have revealed its true underlying structure. Our starting Program Counter Register (0) has been replaced with 00 in its lower byte from the D Register and has effected a jump. This is a great example of ingenuity revealing the underlying nature of a structure, but its without any socially redeeming value (sigh, it takes more instructions and machine cycles to perform)."

Moving right along.

There is a really useful property involved in being given the option to access the Program Counter. It is this: To give programs relocatability (page relocatability) all you have to do is (90, B1, B2, etc.) as part of the initialization process. The high part of the PC moves into the D Register and zips into the registers you wish to initialize. How simple. Elegant even. See Ivan Dzombak's TVT & CHESS on page 3 for an example of this process. The COSMAC magician at RCA can take a bow on that one.

One of the keys to understanding the Elf system lies in understanding the function of the 64 I/O instruction. The 64 instruction does several things at the same time. It sends a quick pulse of electricity to N2. It moves the information in the memory location pointed to by X to the Bus. (With an Elf you get an LED display since N2 and the Bus work in tandem). It also increments the number stored in the Register pointed to by X.

Taking last things first, X is a Register of only four bits (half a byte—a nibble if you prefer). We set X with a SEX instruction. (E5 will set X to the four bit hex number 5). Since there are 16 general purpose registers in all (more registers of the 16 bit variety than any other 8 bit micro), we can point to all of them. These versatile and powerful registers set the 1802 apart from any other micro—and they force one into a different style of thinking.

Each microprocessor is a work of art incorporating a unique style of logical and organizational beauty. With the 1802 there is a certain interesting problem solving ability called into play. You work backwards and are left with a lot of "aha" or "discovery" insights after composing even the simplest of programs. Here is an example of this process. You want to display memory location 02 (hex notation). You put 02 into any one of the 16 registers (let us pick Register 5). F8, 02, A5, F8, 00, B5 does this for us. With a basic 256 memory Elf, you do not need to put 00 into the high part of Register 5. Then, E5 sets the four-bit X to point at Register 5. Now, 64 will display the content of memory location 02 which is A5. To stop the program at this point 00 (IDL) does the trick. So try loading it at starting location 00:

F8, 02, A5, F8, 00, B5, E5, 64, 00

After running this program you will see A5 displayed. If you own a VIP you can check location 0XB5 to see the contents of Register 5. Each time you run the program, the contents of Register 5 will change. (the 0X part = 07, 0B, 0F, for 2K, 3K, and 4K VIP's respectively).

Now what is this about incrementing the stored number in Register 5? If you were to take a picture of Register 5 after it has run the program you would see that it contains 00 03. The way you prove this with an Elf system is to go back and load a new program. You write this new program over the one you just put in—the first locations get changed. Load:

E5, 64, 00

Press reset and run. Surprise. F8 (the contents of location 00 03) are displayed. This proves it . . . Register 5 ~~has been incremented and displayed~~.

This "two pass" process can make a great way to check out the contents of a PROM on an Elf system. Just set the first program to point to the memory you want to see and run. The contents of this location are displayed. Then by putting in the E5, 64, 00 program and pressing RESET, GO, you see the next location contents. Pressing RESET, GO again will give you the next location, and so on...

The world of programming at the machine language level is a magic world. Math becomes logic. Music becomes an ON and OFF series. Language is a series of ASCII code "strings." Graphics and pictures are manipulations of memory bits and "strings." Chess algorithms are especially facinating and beckoning. Chess is no longer a game—it holds the keys to complex decision making ability in computer programs of all sorts.

TVT AND CHESS

LOC.	CODE	COMMENTS
0000	90 B1 B2 B3	Initialize Pointers
0004	F8 33 A3	MAIN
0007	F8 31 A2	STACK
000A	F8 10 A1	INTERRUPT
000D	D3	gosub R3 (MAIN)
000E	72 70	Interrupt Return
0010	22 78	
0012	22 52	
0014	C4 C4 C4	NOP's for sync.
→ 0017	F8 00 A0	Set DMA ptr.
→ 001A	F8 06 B0	
001D	80 E2	Int. Routine
001F	E2 20 A0	
0022	E2 80 A0	
0025	E2 20 A0	
0028	3C 1D	Br. Interrupt
002A	30 0E	Br. Int. Ret.
002C	00 00 00	Stack Area
002F	00 00 00 00	
0033	E2 69	Turn on TV
→ 0035	F8 06 B4	Display Pointer High
→ 0038	F8 00 A4	Display Pointer Low
003B	F8 08 A8	Counter
003E	3F 3E	Wait for INPUT
0040	37 40	depressed and released
0042	6C A6	R6.0 ← Input byte
0044	93 BF	Pointer to Variable High
0046	F8 51 AF	Pointer to Variable Low
→ 0049	F8 02 B5	Points to Table Start High
→ 004C	F8 00 A5	Points to Table Start Low
004F	86 FB 30	Loop until match between
0052	32 5F	Input and ASCII code is
0054	0F FC 01 5F	found—Grab & INC & Replace
0058	15 15 15	Increment the Table displace-
005B	15 15	ment and go back for
005D	30 4F	another try at matchup
005F	F8 05 A7	Set loop counter to 5
0062	45 54	Lookup byte → D & D → M(R(4))
0064	14 14 14 14	More displacement house-
0068	14 14 14 14	keeping stuff; Reg. 4 (8x)
006C	27 87	Dec. and put count in D for
006E	3A 62	Testing; GOTO 62 if D≠00
0070	28 88	Dec. and test Reg. 8 Counter
0072	3A 7A	GOTO 7A if D≠00
0074	F8 08 A8	Reset Counter
0077	14	INC to point to next line
0078	30 82	Go for another Input
007A	F8 27 A7	Set loop counter for top of
007D	24	next char.; DEC R4
007E	27 87	DEC and test
0080	3A 7D	GOTO 7D if D≠00
0082	F8 30 5F	Re-init. Variable
0085	30 3E	GO wait for an input

By Ivan Dzombak

This TV Typewriter recognizes ASCII code with some slight modifications made for convenience. The code for a space is changed from 20 to 30 to make the lookup table more compact. If you want a zero you can use the alphabetic O or you can add a special character to the lookup table, Ø is often chosen. The TVT and chess generator are designed for the do-it-yourselfer. Feel free to modify it and send your suggestions and improvements to QUESTDATA.

When designing characters to add to the graphics ability of the generator, the first step is to draw up a grid of squares 8 (horizontal) by 4 (vertical). From there you translate the on and off locations into their representative hex code. For example, since none of the squares in the top row of the castle are turned on, the corresponding hex code is 00. The second row, or points of the castle, can be represented on graph paper as hex code 2A. See NUMBER PATTERN LOOKUP on this page—0237 is the location of the castle code.

[LETTER PATTERN LOOKUP is on page 4]

****NUMBER PATTERN LOOKUP****

0200	00 00 00 00 00	(SPACE)
0205	06 02 02 02 07	1
020A	0F 01 0F 08 0F	2
020F	0F 01 07 01 0F	3
0214	02 06 0A 1F 02	4
0219	0F 08 0F 01 08	5
021E	03 04 0F 09 0F	6
0223	0F 09 02 04 08	7
0228	0F 09 0F 09 0F	8
022D	0F 09 0F 02 0C	9
0232	FF FF FF FF FF	(WHITE SQ.)
0237	00 2A 1C 1C 3E	(CASTLE)
023C	08 1C 1C 3E 7F	(BISHOP)
0241	08 14 08 1C 1C	(KING)

CHESS

CHANGE:

LOCATION	CODE
003E	30 87

CHESS AND BUFFER PROGRAM:

0087	31 BF
0089	93 B9
008B	F8 98 A9
008E	7B
008F	49 A6
0091	32 96 (00=END)
0093	30 44
0095	7A 30 96

TABLE BEGINS HERE:

0098	3B
0099	3C
009A	30
009B	42
009C	45
009D	30
009E	41
009F	30
00A0	44
00A1	45
00A2	53
00A3	49
00A4	47
00A5	4E
00A6	45
00A7	52
00A8	4F
00A9	46
00AA	30
00AB	43
00AC	48
00AD	45
00AE	53
00AF	53
00B0	30
00B1	50
00B2	49
00B3	45
00B4	43
00B5	45
00B6	53
00B7	30
00B8	3D
00B9	00 [THE END]

P.O. Box 4430, Santa Clara, CA 95054

*****LETTER PATTERN LOOKUP****

LOC.	DATA	
0255	06 09 0F 09 09	A
025A	0F 05 07 05 0F	B
025F	0F 09 08 09 0F	C
0264	0F 05 05 05 0F	D
0269	0F 08 0E 08 0F	E
026E	0F 08 0E 08 08	F
0273	0F 08 0B 09 0F	G
0278	09 09 0F 09 09	H
027D	1F 04 04 04 1F	I
0282	0F 02 02 0A 0E	J
0287	09 0A 0E 09 09	K
028C	08 08 08 08 0F	L
0291	1B 1B 15 11 11	M
0296	19 15 15 13 11	N
029B	06 09 09 09 06	O
02A0	0F 09 0F 08 08	P
02A5	0E 11 15 13 0F	Q
02AA	0F 09 0F 0A 09	R
02AF	07 08 06 01 0E	S
02B4	1F 04 04 04 04	T
02B9	09 09 09 09 06	U
02BE	11 11 11 0A 04	V
02C3	11 15 15 15 0E	W
02C8	11 0A 04 0A 11	X
02CD	11 0A 04 04 04	Y
02D2	1F 02 04 08 1F	Z

To Clear memory before entering TVT Program

Jay Mallin CLEAR PROGRAM

LOC.	CODE	MNEM.
0000	F8 05	LDI
0002	AA	
0003	EA	SEX
0004	F8 00	LDI
0006	73	STXD
0007	30 06	BR

HOW TO RUN TVT PROGRAM:

(1) Enter and Run Jay Mallin Clear Program
(2) Enter Display Program Locations 000-0085
(3) Enter Number Pattern Lookup 0200-0241
(4) Enter Lettern Pattern Lookup on page 4 locations 0255-02D2
(5) Press RESET and RUN
(6) Enter ASCII character 48=H for example
(7) Press INPUT AND SEE CHARACTER DISPLAYED

One of the potential uses to explore for this TVT is its use with other programs. Simple math and alphanumeric displays are possible using this approach. Observe how the pattern generator for light and dark squares works. The pattern to be projected is entered into the TVT at location 0044 and the TVT takes care of projecting the "whatever" and even gets ready for the next letter or number to be displayed. So if the answer to a multiplication program were to enter a character at a time into location 0044, it would be displayed on the TV. What you do is Put Low Register 6 (A6) and jump to location 0044 and the graphic of your choice is displayed. By placing things in a long string in memory and then scanning and putting the string ASCII code by ASCII code, you will have a nice way of seeing results on the TV. An increment, get from the memory buffer (LDN, for example), and a jump to the TVT will then display your buffer memory line. Registers 9, A, B, C, D, E are not used by the program, and can be used for this purpose.

A "mailbox" approach is one possibility for storing the buffer line to go into the TVT. This approach is mentioned on page 55 of ***Chess Skill in Man and Machine*** Edited by Peter W. Frey.

The area displayed by the TVT is given in locations 0017 and 001A (it is page 06). The locations 0035 and 0038 give the starting 06 page location again so that displacement can be figured. The lookup table itself is located on page 02 (locations 0049 and 004C) of memory. Different DISPLAY and LOOKUP table locations are possible, and can be arbitrarily set to your needs.

The castle or ASCII CODE 3B is given just for fun. A good reference for a nice looking set of chess pieces can be found in ***SARGON a Computer Chess Program*** by Dan and Kathe Spracklen [Hayden Book Co.].

So enjoy this alphanumeric and chess generator, and if you have applications ideas and modifications, send them to QUESTDATA so that they can be shared with others. Have fun.

HOW TO RUN CHESS EXPERIMENTER:

(1) Do the first 4 steps of TVT Program
(2) Chang loc. 003E in the Display Program to 30, 87
(3) Enter Program 0087-0097
(4) If you wish a checkerboard of light and dark squares enter 30, 3A, 30, 3A, 30 3A, 30, 3A, 3A, 30, 3A, ETC. End Pattern with 00!
(5) For a fun challenge enter the BUFFER TABLE given on page 3 loc. 0098-00B9
(6) After choosing (3) or (4) of the above Press: RESET, RUN

WHY CHESS IS BECOMING MORE THAN A GAME

[Note: For further reading on the subject of chess automations, Floyd L. Oats highly recommends the October, November, and December issues of Byte Magazine *(1978). Another source of information is* Chess Skill in Man and Machine, *edited by Peter W. Frey, published by Springer-Verlag (New York, 1977). Floyd L. Oats is nearing completion of a chess playing algorithm for the COSMAC. This program will be made available to you in some form or other as soon as it is completed. Stay tuned.]*

By Floyd L. Oats

If one could create a chess playing machine capable of playing above the Master level, then it would seem that the very essence of the human intellectual process had been captured. That statement reflects the opinion held by many researchers in the fields of psychology and artificial intelligence, and explains why many of these people are actively involved with computer chess programming although they might have little interest in the game of chess in itself.

The selection of a move by a human player is performed by a complex interaction of perception, chess knowledge and experience, along with practically every intellectual faculty the human player has. Stated in its simplest terms, when a human chess player selects a move, he has solved a complex problem. Complex, not only because of the infinite lines of play the game may assume, but also because there is no exact or perfect solution (except in a few special board positions).

Since the human brain and nervous system can be viewed as machine-like in nature, researchers believe that it should be possible to imitate them with computers. Such a super chess playing machine, some feel, would be a mechanized model for the human mind and could open new insights for those involved in psychology and artificial intelligence. The concepts revealed by such a device could be applied in other complex problem-solving endeavours, including economic, social and educational systems. For example, could the computer salvage the sagging dollar? The field of robotics could also profit from such knowledge.

Since most of us aren't researchers, why should we be interested in computer chess? If you understand the basis for chess algorithms, you are in a position to accept the challenge of creating a chess playing program. Such programs employ virtually every programming "trick" in the book from simple movement of data from place to place, through and including complex sorting algorithms.

The demands placed on a chess program in terms of speed, efficiency, effectiveness and memory utilization create a seemingly endless line of trade-offs. These trade-offs force the programmer to make decision after decision while maintaining a good mental picture of how the program functions as a whole. It is truly a noteworthy accomplishment for a programmer to write a chess program capable of challenging the average human player.

Once you get this program written, what do you really have? You have an opponent who is always willing to play, who doesn't toot his horn when he wins, and doesn't make excuses when he loses. If he loses his queen, he does not resign to a disinterested passive role as humans often do, but continues to strive for a win in spite of overwhelming odds. Nor does he relax in anticipation of easy victory after robbing your material. Most chess programs play for one purpose—to WIN!

Another reason people write chess programs is for competition. There are several computer chess tournaments held every year, some exclusively for microprocessors and some for anything one has the courage to enter into play. The current world champion chess program is Northwestern University's CHESS 4.7. Rest assured that we aren't quite ready to tackle him with a microprocessor. To the best of my knowledge, there is no world title for microcomputer chess programs (although I would like to see one). The spirit of competition has stimulated the development of new ideas in chess programming but the secrecy surrounding such projects tends to discourage free distribution of these concepts for obvious reasons.

The first document describing a practical chess playing algorithm appeared in 1950 and was prepared by the English mathematician, Claude Shannon. He proposed a tree-searching algorithm in which the trunk of the tree, or the "base node" as it is often called, is the current board configuration. Each legal move which the computer's pieces can make from the base node leads, in turn, to another node. From each of these nodes, there is a set of branches representing the opponents set of counter-moves and each of these lead to another node.

The Shannon program will generate every possible combination of moves, counter-moves, counter-counter moves, etc., to a certain depth in the game tree, and then will perform a static board evaluation on each final position or "terminal node." The board evaluation function is the heart of the chess program. It is here that the computer performs an operation that human players never attempt—it quantitizes the board position! The computer requires numbers to work with, so the program builds a number which represents (a) who has the advantage, computer or opponent and (b) the exact magnitude of this advantage. A common scheme is to let positive numbers represent the computer's advantage and negative numbers represent the opponent's advantage. The absolute value of the number reflects the magnitude of the advantage. This function should yield an accurate picture of the true situation since this is the only source of numerical data upon which the computer makes its move.

Since there can be tens of thousands of terminal positions (depending upon how deep the computer goes into the game tree), it is imperative that the evaluation be done quickly in order for the machine to choose a move within a reasonable length of time. Since speed of evaluation and accuracy of evaluation are mutually exclusive requirements, we are faced with a very serious trade-off decision. In general, searches which go deeper into the tree use more primitive evaluation functions.

In computer chess, each half-move is called a "ply," so a complete move is made up of two plies, one from each participant. The base position is called ply zero and represents the actual board configuration that has been reached in the game. It is from this position that the program will "mentally" move pieces around and evaluate resulting board positions in order to determine which legal move seems the best.

Going back to the tree concept, first we have the actual board position existing at a certain point in the game. The lines leaving this box would be branches representing the set of legal moves which the computer may execute from this position. We will restrict the number of moves from each node to three for illustrative purposes and, to further simplify the discussion, we will assume that all pieces are at home position on the base node. The computer will execute the first move of the game as white.

We will restrict the tree search to a fixed depth of three plies. The computer will begin by executing a move from ply zero which, of course, will be a white piece move. Let's say that he starts with the P-K4 branch. Next, the program generates a legal move of a black piece from ply one, say, the P–K4 branch from ply one. Sitting at ply two now, he generates another legal move of a white piece and goes to ply three, let's say the leftmost block at ply three. Realizing that the depth limit has been reached, this position is declared a "terminal note" and the static evaluation is performed.

The numbers in the ply three boxes indicate values calculated by the static evaluation function for each ply three node. The leftmost block at ply three is evaluated as zero, the move is un-made, and this zero is returned to the leftmost block at ply two. We now generate the next legal move from ply two which takes us to the second box in ply three for an evaluation of -1. We un-make that move, take the evaluation back to ply two and generate another move from —continuing in this fashion until all legal moves of all white pieces from the leftmost ply two node have been processed (in this case three moves). The act of taking the evaluation back to ply two is called backup.

Of the three static evaluations for the leftmost ply two node, we will take the most positive evaluation of that particular ply two node. Next, we will un-make the opponents P–K4 move and go back to ply one from which we will execute the opponents N–QB3 move. We are now at the second node of ply two and we will repeat the set (possibly a DIFFERENT set now) of legal moves into ply three. This time we return a +8 to the ply two node. We go back to ply one and pick up the N–KB3 node and repeat the above actions.

Eventually, all the moves from ply one are exhausted and we return the most NEGATIVE value from the ply two nodes, since these are the opponents moves. The computer assumes that the opponent will attempt to minimize the value of the board positions and will choose the appropriate move with the most negative evaluation (negative numbers favor the opponent).

As a result, a set of values are backed up into ply 0. Since the computer moves from ply zero, it will select the move which returns the most positive evaluation. The general rule for backing values up the tree is to maximize when backing into an even ply and minimize when backing into an odd ply. This procedure is commonly used even in state-of-the-art computer chess programs and is called a depth-first mini-max procedure.

The depth-first comes from the fact that we go to the maximum depth in the game tree BEFORE any evaluations are made. The mini-max, of course, refers to the manner in which the values are backed up. Incidentally, if the values shown in Figure 1 were to appear in a real chess program, the computer's chosen move would be P-K4.

The three ply search of Figure 1 requires 27 board evaluations, one for each of the ply three nodes. If there had been 20 moves from each of the ply zero, one and two nodes three would have been 20 x 20 x 20 = 8000ply three board evaluations! Experts agree that the average number of moves from a ply is 38, so the average three ply search would require about 60,000 static board evaluations. When every possible move from each ply is considered by the program it is called a full-width or brute-force search. Shannon labelled it "type A strategy."

The primitive computers of the late forties simply could not hold up against the kind of mathematics we have just seen, so Shannon described a type B strategy which seemed more feasible for computer application. It is known that there are never more than two or three plausable moves from any board position. If the computer could pick, say, five plausable moves at each ply, then a four ply search would require only 625 evaluations—a very manageable number.

Many modern programs, especially microprocessor programs, employ the type B strategy. The number of plausable moves selected at each ply is rarely less than 16, however. As a general rule, programs which use the type B strategy do not fare well against those employing a full-width search.

The static evaluation function dictates the "behavior" of the chess program. By far the heaviest term in the evaluation function is the "net material balance." A common means of defining material value is to assign a standard pawn a value of 100 points and specify other values in terms of this standard unit. A knight is usually 325 points, a bishop 350 points, a rook is 500 points and the queen is 900 points. A standard pawn is defined as one of the six non-center pawns, sitting on its home square. Since there is a good correlation between material and chances of winning, and since material value is extremely simple to represent and keep track of, it makes good sense to cause the evaluation function to be material-heavy.

There are other factors considered by the evaluation function such as piece mobility, center control, king safety, and piece cooperation. Piece mobility is often computed by adding all the legal moves that can be executed from the board position under evaluation. Center control is often included in the piece mobility term by giving bonus points for each center square which a piece can move to or through (i.e., under attack). The king safety term will consider such things as pawn structure around the king and squares adjacent to the king which are under attack by enemy pieces. To discourage shuffling the king around aimlessly in relatively quiet board positions, there is usually a fee exacted for moving the king a single square. This fee is waived if the king is moved two squares in order to encourage castling. A typical value for this fee is 20 points.

Piece cooperation is the term which tends to give each chess program its own individual tendencies and behavior. This is where the chess programmer scratches his head and begins making the big decisions. Probably the toughest portion of this routine involves exchange evaluation, which must be done when a piece is under attack by both sides. By examining the number and the material value of all attackers of each color, the program can determine whether the piece under attack can be profitably captured by the enemy. The program MUST figure this into the evaluation function in order to prevent gross blunders. A substitute for this particular term is a deeper search with material balance analysis, but this is out of the question in microprocessors programs—we simply don't have the time.

Another factor in the cooperation term discourages the purposeless moving of a piece back and forth by docking the piece typically 30 points for being moved twice in succession. This usually discourages moving the piece twice in a row unless it is en route to a powerful and strategic post. Another factor is piece development, whereby non-king pieces are penalized for residing on the friendly back rank. The penalty for queens and rooks is generally about half of that for a knight or bishop in order to develop minor pieces first.

As the game wears on, certain values begin to change. As the end game begins to unfold, for example, the king should come out of hiding and begin actively cooperating with his remaining pieces. During end game play, the king should be penalized for back rank and edge occupation (constitutes a bonus in early game play). As pawns approach the back rank, their material value increases, a typical value being 200 points for a pawn on the seventh rank.

Bonuses for doubled rooks and king tropism are usually increased during the end game. In microcomputers, the end game is generally detected by counting the number of moves that have been made in the game, something on the order of 35 moves indicates that the end game has arrived. This is not a foolproof method for detecting the end game.

It is really the programmer and his idiosyncrasies that control the movement of the chess pieces. He selects the factors which will be considered for evalution process and the relative weight of each chosen factor. He can experiment with his creation by changing the amounts of bonuses and penalties for the various factors. Most important of all, he makes the trade-off decisions for each factor considered by the evaluation routine: Is this element worth the computer time required to consider it? The programmer has total control!

All of the non-material terms incorporated into the evaluation function including center control, piece cooperation, etc. come under the heading of "positional terms." This means that there are two kinds of terms in the evaluation routine, material and positional. If you delve into chess programming very deeply, you will soon discover that there are ways of cutting down the size of the game tree (pruning). Also, there are ways of bypassing the grueling positional terms in some board positions (scrapping). The justification for these procedures is rather lengthy and space will not permit us to discuss them, but they are mentioned for the sake of completeness.

The algorithm we have just described is not very human-like in its move selection. It is known that humans can play a good game of chess without considering every possible move down to so many plies in the game tree, so it follows that computers should be able to do the same. There are those who are making progress with programs designed to select moves strictly on the basis of a single but very lengthy static evaluation of the existing board configuration. While these programs are more human-like in their approach to move selection, few of them have abilities above the advanced novice level. But who knows, maybe someone will discover a "magic" evaluation procedure.

The ability of computers using the Shannon-type strategy is rated at about thirteen to fourteen hundred on the USCF (United States Chess Federation) scale. In spite of the perfect mathematical precision with which the rules and goals of chess are defined, even the largest and most powerful computers in the

(Continued on page 12)

NEW IMPROVED TARGET GAME WITH JUMPING MAN

By Jack Krammer

This new target game runs in 1K of Elf memory and contains the following new secret ingredients:

- On screen scoring of shots and hits. The game stops after the completion of the 15th shot.
- A video character (person) that provides animation by throwing the shot (on player command) and jumps spread eagle if the target is properly hit.
- A target that looks like a right hand bracket with a handle. If the target is hit in the cap opening it stops its downward motion and displays a broken handle. A hit of either the top or bottom arm of the target will remove that element, but is considered a (close) miss and does not score as a hit.

The first three bytes (M(0000, 1, 2) are reserved for a long branch instruction for those systems with a resident operating system. For those who do not have a resident operating system, there is a short but effective program starting at M(003) to M(0011), a total of 15 bytes, which allows you to load and or read memory. If you wish to use this loader program do the following:

- Put the high order address byte at M(0005) and the low order address byte at M(0008).
- Put a short branch instruction (30 03) at M(0000) and run the program.
- When the memory protect switch is OFF you are writing into locations. When the memory protect is ON you can read the contents of memory. The data (either read or write) will appear on the display and the program will point to the next higher addres in memory, ready to repeat the desired operation.

The program keeps score and displays the results of 'SHOT' or 'HIT' in the following manner. Register R(4) is initialized to 00 00 for the 'HIT' and 'SHOT' scores. The appropriate register (R(4).1 or R(4).0) is put into R(7).0 which then points into the 'TABLE.' In this condition R(7) points to M(0200) which is the start of the number character 0. The scoring of the game or 'SCORE DISPLAY' subroutine, takes five bytes from 'TABLE' and puts them into the appropriate locations in the display by R(8) and the character 0 is displayed. For each 'SHOT' or 'HIT' the program adds 5 to the contents of the appropriate half of R(4), thereby providing direct pointing information into 'TABLE' for display of the next number. Register R(8) initially points to M(0301) for 'SHOT,' M(0304) for 'HIT,' M(03C8) for video character and M(03BF) to erase the broken target. The 'SCORE DISPLAY' subroutine places the five bytes from 'TABLE' one below the other in a vertical format in the display and the number or character is formed.

Using the above technique it is possible to generate a table of pictures and with the subroutine under an animation program control develop a computer video movie, complete with titles, etc.

The "TV GAME" starts at LOCATION 0012. By placing a short branch instruction (30 12) at LOCA-ATION 0000 and hitting RUN, the fun begins. Pressing the INPUT button causes the video character to throw the shot at the target.

PURPOSE	LOC.	CODE	COMMENTS
	† 0000	*30 12*	† RESERVED FOR SHORT OR LONG BR.
'Read/Write	0003	E1	R(1) Data Pointer
Memory'	0004	F8 ** B1	Hi Order address to R(1).1 ** SEE
	0007	F8 ** A1	Lo Order address to R(1).0 TEXT
	000A	3F 0A	A loop to wait for INPUT to be pressed
	000C	6C	Keyboard to M(R(1)), see text
	000D	64	M(R(1)) into DATA DISPLAY; R(1)+1
	000E	37 0E	Wait for INPUT to be released
	0010	30 0A	Return to loop
'Clear	0012	F8 03 B1	03FF into R(1) top address of
Display'	0015	F8 FF A1	Display Page
	0018	E1	R(1) Data Pointer

Puts 00 in	0019	F8 00 73	(loop) 00 into D into M(R(1)); R(1)-1
all locations	001C	81	R(1).0 into D
of the display	001D	3A 19	Return to loop if D≠00
Page	001F	73	(D=00) into M(R(1))
'Initialize'	0020	F8 00 B1	
	0023	F2 A3 B4	
	0026	A4 AF	
	0028	F8 01 B3 BA	
	002C	F8 03 B5 B8 BE	
	0031	F8 40 A1	
	0034	F8 F0 A2	
	0037	F8 B9 AA	
	003A	F8 02 BD	
	003D	D3	
'Display	003E	72 70	
Refresh'	0040	22 78 22 52	
	0044	C4 C4 C4	
	0047	F8 03 B0	
	004A	F8 00 A0	
	004D	80 E2	
	004F	E2 20 A0	
	0052	E2 20 A0	
	0055	E2 20 A0	
	0058	3C 4D	
	005A	30 3E	
'Program	0100	F8 02 B7 B9	R(7) 'Table' & R(9) 'Score Display Hi addr.
Begins'	0104	84 A7	R(4).0 ('Shot') into D into R(7).0
	0106	F8 01 A8	01 into D into R(8).0 'Shot' Display address
	0109	F8 50 A9	50 into D into R(9).0 'Score Display' Lo address
	010C	D9	GOTO 'Score Display' Subroutine
	010D	F8 50 A9	Reinitiialize 'Score Display' Lo address
	0110	94 A7	R(4).1 ('HIT') into D into R(7).0
	0112	F8 04 A8	04 into D into R(8).0 'HIT' Display address
	0115	D9	GOTO 'SCORE DISPLAY' subroutine
	0116	F8 50 A9	(See M(010D))
	0119	E2 69	R(2) Data Pointer, Turn on TV chip
	011B	F8 00 A6	Initialize 'SHOT SPEED' counter
	011E	F8 C0 AE	Initialize 'PROJECTILE' position
	0021	F8 80 5E 7A	Locate 'PROJECTILE'; 0 into Q.
	0025	F8 60 A7	60 into D into R(7).0 Points to 'READY'
	0028	F8 C8 A8	C8 into D into R(8).0 'Player' display address
	002B	D9	GOTO 'SCORE DISPLAY' subroutine
	002C	F8 50 A9	(See M(010D)
	002F	F8 02 BB	02 into D into R(B).1 Initialize 'Projectile' speed
	0032	2B 9B 3A 32	Decrement 'Projectile' speed counter to zero
	0036	16 86	Increment 'SHOT SPEED' counter
	0038	FD 05 3A 8A	If 'Shot Speed' counter not 05 GOTO M(018A)
	003C	F8 00 A6	Initialize 'shot speed' counter
'Move	013F	8F 3A 6B	R(F).0 into D, D≠00 GOTO 6B
Target'	0142	F8 EF A5	EF into D into R(5).0
	0145	F8 00 55	00 into D into M(R(5))
	0148	F8 F7 A5	F7 into D into R(5).0
	014B	F8 00 55	00 into D into M(R(5))
	014E	F8 FF A5	FF into D into R(5).0
	0151	F8 00 55	00 into D into M(R(5))
	0154	F8 07 A5	
	0157	F8 C0 55	
	015A	F8 0F A5	
	015D	F8 7F 55	

	0160	F8 17 A5	
	0163	F8 C0 55	
	0166	F8 1D AF	Reset X29 COUNTER
	0169	30 2F	GOTO M(012F)
	016B	2F	Decrement X29 COUNTER
	016C	F8 03 AD	Set X3 COUNTER
'Move Target	016F	85 FC 08 A8	R(5).0 into D+08 into R(8).0
Down One	0173	05 58	M(R(5)) into M(R(8))
Line On	0175	85 FF 08 A5	R(5).0 into D-08 into R(8).0
Display'	0179	2D	Decrement X3 COUNTER
	017A	8D 3A 6F	X3 COUNTER ≠0 GOTO 'Move Target'
	017D	85 FC 08 A5	R(5).0 into D+08 into R(5).0
	0181	F8 00 55	00 into D into M(R(5))
	0184	85 FC 18 A5	R(5).0 into D+18 into R(5).0
	0188	30 2F	
	018A	31 A5	Q=1 then GOTO 'Move Shot'
	018C	3F F6 7B	Input not pressed GOTO ('End Game'); Else 1 into Q
'Puts Thrown	018F	F8 65 A7	65 into D into R(7).0 points to 'Thrown'
Figure On	0192	F8 C8 A8	C8 into D into R(8).0 'Player' Display address
Display'	0195	D9	GOTO 'Score Display' subroutine
	0196	F8 50 A9	(See M(010D))
'Increment	0199	84 FC 05 A4 A7	R(4).0 ('Shot') into D+05 into R(4).0 & R(7).0
'Shot' No.	019E	F8 01 A8	01 into D into R(8).0 'Shot' Display address
	01A1	D9	GOTO 'Score Display' Subroutine
	01A2	F8 50 A9	(See M(010D))
'Move	01A5	0E F6 5E 3B AE	M(R(E)) into D, SHR, D into M(R(E));DF=0 Br. AE
Shot'	01AA	1E	Increment R(E)
	01AB	F8 80 5E	80 into D into M(R(E))
	01AE	8E FF C7	R(E).0 into D-C7 into D
	01B1	3A 2F	D≠00 GOTO M(012F)
'Target	01B3	85 FF 08 5A	R(5).0 into D-08 into M(R(A))
Hit?'	01B7	8E FF 00	
	01BA	3A EC	D≠00 GOTO 'Remove Residue Shot'
"Puts 'spread	01BC	F8 6A A7	6A into D into R(7).0 Points to "spread eagle'
eagle' on	01BF	F8 C8 A8	C8 into D into R(8).0 'Player' Display address
Display	01C2	D9	GOTO 'Score Display' subroutine
	01C3	F8 50 A9	(see M(010D))
	01C6	94 FC 05 B4 A7	R(4).1 ('Hit') into D+05 into R(4).1 & R(7).0
	01CB	F8 04 A8	04 into D into R(8).0 'Hit' Display Address
	01CE	D9	GOTO 'Score Display' Subroutine
	01CF	F8 50 A9	(See M(010D))
'Broken	01D2	F8 15 5E	15 into D into M(R(E))
Target'	01D5	F8 40 BC AC	Reset 'Show Broken Target' Counter
	01D9	2C 9C	Decrement R(C), R(C).1 into D
	01DB	3A D9	D≠0 GOTO M(01D9), ELSE
	01DD	F8 70 A7	70 into D into R(7).0 Points to 'Clear'
	01E0	F8 BF A8	BF into D into R(8).0 Points to Target
	01EB	D9	GOTO 'Score Display'
	01E4	F8 50 A9	(See M(010D))
'Delay'	01E7	F8 05 AF	05 into D into AF Resets x29 counter
	01EA	30 F5	GOTO 'End Game'
'Remove	01EC	F8 03 BB	03 into D into R(B).1
Residue	01EF	F8 C7 AB	C7 into D into R(B).0
Shot'	01F2	F8 00 5B	00 into D into M(R(B))
'End Game'	01F5	84 FC B5	R(4).0 ('Shot') into D-B5 into D
	01F8	3A 1E	If D≠0 GOTO M(011E), Else
	01FA	00 30 FA	Endless Loop

'Table' and More Target Game with Jumping Man on **Page 11**

'Table'	0200	07 05 05 05 07	0
	0205	02 06 02 02 07	1
	020A	07 01 07 04 07	2
	020F	07 01 03 01 07	3
	0214	04 05 07 01 01	4
	0219	07 04 07 01 07	5
	021E	07 04 07 05 07	6
	0223	07 01 01 01 01	7
	0228	07 05 07 05 07	8
	022D	07 05 07 01 01	9
	0232	27 65 25 25 77	10
	0237	22 66 22 22 77	11
	023C	27 61 27 24 77	12
	0241	27 61 23 21 77	13
	0246	24 65 27 21 71	14
	024B	27 64 27 21 77	15

QUESTDATA
P.O. Box 4430
Santa Clara, CA 95054

Publisher . . . . Quest Electronics
Editor . . . . Bill Haslacher
Technical Coordinator . . . . . . Jane McKennon
Circulation . . . . Laurie Buzzell
Proofreading . . . . Ken Brown

'Score Display'	0250	F8 05 AD		Set X5 Counter
	0253	E7		R(7) Data Pointer
	0254	72	Loop	M(R(7)) into D, R(7)+1
	0255	58		D into M(R(8))
	0256	88 FC 08 A8		R(8).0 into D+08 into R(8).0
	025A	2D		Decrement R(D), X5 Counter
	025B	8D 3A 54		R(D).0 into D, D≠0 GOTO (M(0254)) Loop
	025E	E2		R(2)
	025F	D3		Return to MAIN PROGRAM

'Characters'	0260	90 FE 3A 38 6C	'Ready'
	0265	10 FE BA 38 6C	'Thrown'
	026A	92 FE 38 BA 44	'Spread Eagle'
	026F	00	(Not Used)
	0270	00 00 00 00 00	(Zero's to Clear Broken Target)

(Continued from page 7)

world cannot play at the master level, and cannot play consistently at the expert level. The search goes on for that secret ingredient contained in the human mind that man has not yet been able to impart to his machines, and chess programming is one of the avenues for this research.

Consider that the average number of plies searched by the big machines is five, and that the average number of moves from a board position is 38. It would require a 38-fold increase in computer speed to yield one extra ply of tree-searching ability. In numerical terms, this amounts to a 3800% increase in speed to render a mere 20% increase in general playing strength. The mathematics of Shannon's approach don't look much more promising than they did thirty years ago. But don't get the idea that a computer cannot challenge the average human player, Because IT CAN!

The computer never overlooks pieces that are under attack and not protected, possible check sequences, pieces that can be profitably captured, profitable exchanges, etc. While the computer has no long-range plan, it is adept at finding holes in your plan and using these holes to harass you. Nor will it fail to "see" any blunders you might commit as a human opponent sometimes will. The computer's short range precision seems to make up for its lack of long-range foresight.

There are several chess playing computers available to customers, each capable of playing a better than average game. Some are the type A strategy and some use the type B. The details of the algorithm vary from one machine to the next, and some are capable of defeating others. BEWARE of those who claim that their microprocessor searches five moves (ten plies!) ahead or better. They are using a type B strategy with an extremely small number of plausable moves from each ply and can usually be defeated by a full width four ply search. Experts have accurately described plausible move generators as well-meaning but stupid.

Using something on the order of eight or fewer plausable moves per ply invites disaster. Brilliant sacrifices, for example, incur immediate and massive loss of material, so they tend to be rejected early by plausible move generators—without any further consideration. The full width strategy will not abandon a line of play simply because of early losses. There is one common denominator among all the chess programs which play reasonably well against human players—when you knock off the hub caps and peel off the paint, you will find the Shannon tree-search concept at the base of the program structure. After 30 years Shannon is still the modern state-of-the-art!

COSMAC CLUB COSMAC CLUB COSMAC CLUB COSMAC CLUB COSMAC CLUB COSMAC

9

QUESTDATA
P.O. Box 4430
Santa Clara, CA 95054

ADDRESS CORRECTION REQUESTED

BULK RATE
U.S. Postage Paid
QUEST
Electronics
Permit No. 549
Santa Clara, CA